VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Klasifikace obrázků založená na rysech

Image Classification Based on Features

2015

Ondřej Maceček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Ondřej Maceček**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Klasifikace obrázků založená na rysech Image Classification Based on Features

Zásady pro vypracování:

Cílem práce je implementovat algoritmy pro detekci vybraných rysů v obrázcích a jednoduchý systém na vyhledávání obrázků na základě rysů zvolených uživatelem. Součástí práce je provedení průzkumu existujících přístupů a implementace aplikačního prostředí pro experimenty v jazyku C#.

1. Průzkum a popis existujících přístupů.
2. Návrh a implementace vybraných nebo vlastních metod.
3. Návrh a implementace počítačové aplikace pro provádění experimentů.
4. Návrh, realizace a hodnocení experimentů.

Seznam doporučené odborné literatury:

[1] Hasitha Bimsara Ariyaratne, Koichi Harada: Content based Image Retrieval using Spatial Relationships between Dominant Colours of Image Segments. VISAPP 2010:184-189.
[2] Zdenek Horak, Milos Kudelka, Vaclav Snasel. FCA as a Tool for Inaccuracy Detection in Content-Based Image Analysis. Granular Computing (GrC), 2010, 223-228.
Další dle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Miloš Kudělka, Ph.D.**

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 5. května 2015 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2015 .....

Poděkování

Mé velké poděkování patří všem účastníkům průzkumu, kteří mi byli nápomocni při získávání výsledků pro vývoj systému, a dále děkuji vedoucímu diplomové práce Mgr. Miloši Kudělkovi, Ph.D., za jeho rady a čas, který mi věnoval při řešení dané problematiky.

Abstrakt

Cílem této diplomové práce je vytvoření vhodného obrázkového vyhledávacího systému založeného na klasifikaci dle vybraných rysů. První část je věnována již existujícím metodám nápomocným pro celkový rozbor obrázku. Díky tomu lze pak přiřadit rysy pro další vyhledávání uživatelem.

Podstatnou částí textu je také vlastní implementace, kde jsou popisovány veškeré postupy, jak teoretickou formou, tak praktickou. Setkáme se zde například s tvorbou anotované sady obrázků, vývojem barevných odstínů pro hledané barvy, rozdělením obrázků do určitých kategorií, dle kterých je lze třídit. Dále poznáme postup segmentace obrázku spojený s přiřazováním jeho částí k triviálním objektům. V neposlední řadě vyhodnotíme a uložíme výsledné rysy do databáze.

V poslední části je popsán experiment vyhodnocující kvalitu vlastní implementace na anotované sadě obrázku, dle které lze uvážit změnu metody či určité vylepšení.

Klíčová slova: Klasifikace, Rys, Segmentace, Anotovaná sada obrázků, Barevný prostor.

Abstract

The aim of this master's thesis is to create appropriate image searching system based on classification according to specific features.

The first part focuses to existing methods that are useful for total image analysis that helps us to assign features for next user's searching.

The main part of a thesis is my own implementation. There are described all processes theoretically and practically. Part of it is annotation data set creation, color shades development for searched colors and image division into specific categories for their sorting. We get familiar with image segmentation connected with assigning of its parts to trivial objects. Last but not least, we evaluate and save final features into database.

In conclusion is described an experiment that measures own implementation quality on annotation data set where we can consider some method changes or improvements.

Keywords: Classification, Feature, Segmentation, Annotation data set, Color space

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

21. stolení se vyznačuje velkým rozvojem automatizace a digitálního zpracování. Počítače začínají řídit téměř vše a tím se zvětšuje množství dat, které nás přímo zahlcují. Tyto data je nutné uchovávat na různých uložištích, odkud musí být k dispozici pro jejich následné použití. Se vzrůstajícím počtem množství dat, nám stoupá náročnost jejich vyhledávání. Proto jsem se v mé diplomové práci zaměřil právě na problematiku vyhledávání dat, konkrétně obrázků dle různých rysů.

V teoretické části této práce si představíme barevné modely typu RGB a HSL a matematický převod mezi nimi. Dále zde bude věnována patřičná část tvorbě barevné palety i s jednoduchým experimentem, který poukáže na rozdílné lidské cítění barev. Tuto informaci patřičně využijeme pro tvorbu barevné palety, kterou uživatelům systému nabídneme jako klasifikátor pro vyhledávání obrázků. S barevnou paletou se pojí také metody, které řeší přiřazování odstínů barev z palety. Bude se jednat o přiřazování k nejbližší barvě za pomocí vektorové vzdálenosti, tvorbu oblasti, kde se barva nachází aj. Daná kapitola je věnována i problematice segmentace obrázku, díky níž rozložíme obrázek na logické části, za pomocí kterých dále zjišťujeme nové klasifikátory na jejich vlastní hledání. V závěru této kapitoly jsou popsány metody na rozpoznávání triviálních objektů typu čtverec, obdélník, kruh a další.

V části vlastní implementace poznáme jednotlivé postupy, kterými lze dosáhnout efektivního vytváření rysů v obrázku. Je zde i postup tvorby vlastní anotované sady obrázku, pro kterou budou v experimentu zveřejněné i výsledky úspěšnosti detekce. Podstatnou část práce zabere experimentální průzkum na vývoj barevných odstínu z palety. Seznámíme se zde s problémem rychlosti zpracování obrázku, jehož řešení bude podrobně popsáno postupem a metodami nezbytnými pro celkové zrychlení. Dále si představíme segmentaci obrázku a přiřazování tvarů k jednotlivým objektům v obraze. Dospějeme zde k závěru, že segmentace hraje největší roli v konečné fázi, kde se definují rysy přiřazované obrázku.

Poslední kapitola je věnována vlastní implementaci samotného programu na detekci rysů. Pro lepší pochopení je zde znázorněn i třídní diagram.

Výsledkem mé práce bylo prozkoumat existující metody a vhodně je užít s metodami vlastními za účelem vytvoření vyhledávacího systému pracujícího na principu klasifikace obrázků vybraných rysů obsažených v něm. Díky programu již nebude nutné obrázky manuálně třídit podle různých názvů nebo kategorií uživatelem. Všechny budou uloženy v jedné databázi, odkud budou následně načítány dle jednoduchých klíčů - obrázkových rysů. Způsob vyhledávání ocení zejména fotografové, designéři, novináři ale i tvůrci webových prezentací.

2 Průzkum a popis existujících přístupů

V této kapitole nalezneme obecný popis metod, specifikací a známých modelů, které již existují a jsou stavebními kameny pro vytvoření obrázkového vyhledávacího systému, jehož hlavní funkcí bude rozpoznávání jednotlivých obrázků mezi sebou, což umožní uživateli vyhledávat jednotlivé obrázky. Výsledky jednotlivých modelů mohou poskytovat vhodné nebo méně vhodné výsledky a proto je třeba při sestavování systému, promyslet jejich klady a zápory, navrhnout vhodné alternativy jejich využití a mnoho dalších aspektů.

Na konci této kapitoly budeme schopni vytvořit systém, který dokáže detekovat triviální rysy, což mohou být: název, velikost, poloha obrázku (vertikální, horizontální) nebo i složitější, což může být barevná výplň, rozlišování základních objektů aj. Všechny tyto rozpoznávací rysy podrobně rozebereme v další kapitole a ukážeme si jejich vzájemné kombinace, které umožní komplexní vyhledávání.

Nejzákladnějším vyhledávacím rysem jsou však barvy, protože právě s nimi pracujeme všichni každý den při práci s počítačem nebo jiným zařízením, ať už prostřednictvím monitorů nebo jiných displejů. Obrázky obsahující barvy, jsou nejčastěji ve formátu JPG nebo PNG.

2.1 Barevné modely

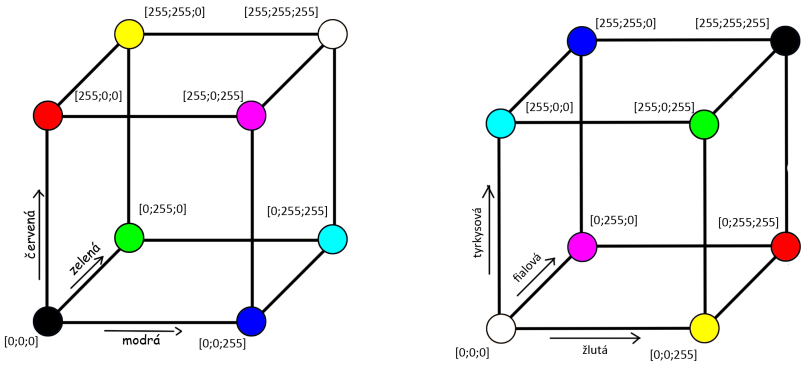
Každý barevný model má specifické vlastnosti, které představují různou světelnost, sytost, barevný tón aj. Tyto vlastnosti se dále určitým způsobem a v různém poměru míchají. Tímto mícháním vlastností dosáhneme obrovské škály barevných odstínů, kterou známe z reálného světa. Dostaneme se tak často k takovému počtu, jež ani lidské oko nedokáže rozeznat. Počty odstínů se v každém barevném modelu liší, a proto je nutné pečlivě promyslet, s jakými barevnými modely budeme chtít pracovat.

Jak je známo, tak lidské oko vnímá barvu působením pomocí tří základních barevných receptorů na sítnici. U většiny lidí jsou tyto receptory nejvíce citlivé na tři základní barvy, resp. jim odpovídajícím vlnových délkám. Jedná se o barvu červenou (red), zelenou (green) a modrou (blue). Tyto informace byli základem při vytváření barevného modelu RGB, který je používán na klasických monitorech, kde jsou všechny odstíny vypočítávány právě mícháním těchto tří základních barev. Ostatní barevné modely si také dále představíme. Informace o barevných modelech jsem čerpal z [2] kde jsou popsány i další modely.

2.1.1 RGB a CMY

Mezi nejvyužívanější barevné modely patři bezesporu RGB, protože se v tomto modelu pracuje s barevnými filtry digitálních fotoaparátů a většina fotografií je v tomto formátu také ukládána. RGB model je asi nejpřirozenější způsob, jak vyjádřit to, co lidské oko vidí. V podstatě nám říká, jak moc je působeno na R – červený receptor, G – zelený a B – modrý, což je lidské oko. Sada třech osmi bitových čísel ve výsledku určí nejen barvu ale také intenzitu světla.

Každá barva z RGB je tedy ukládána jako osmi bitové číslo, neboli 256 hodnot v intervalu <0 − 255>. Libovolná barva je zadána 24 bity vůči třem základním barvám. Ve výsledku máme tedy 256^3 různých barevných odstínu nazývaných True Color, které když přepočteme do čísel, tak výsledek je přesně 16 777 216 barev. V tomto čísle nachází největší slabinu RGB model. Podle studií totiž dokáže lidské oko rozpoznat jen cca 10 miliónů odstínů. Různé studie se mohou lišit, avšak číslo 10 milionů zatím nepřekročily. RGB však obsahuje téměř o 7 milionů barevných odstínů více, a to představuje velký problém u přiřazování True Color k barvám hledaným naším systémem. Bohužel práci s RGB modelem se vyhnout nedá, takže bude nutné tento model převést na model jiný, čímž docílíme nižšího počtu odstínů.



Obrázek 1: RGB barevný model, CMY barevný model

Jak lze vidět na obrázku 1, RGB model lze zobrazit jako prostorovou krychli, kde v počátku RGB modelu leží barva černá, vyjádřena vektorem $\vec{v}\{0;0;0\}$, a v protilehlém vrcholu barva bíla, vyjádřena vektorem $\vec{v}\{255;255;255\}$. Jakýkoliv jiný poměr míchání barev nám určí rozdílnou barvu, alespoň co se číselné informace týče. Pro lidské oko se totiž nepatrné změny projevit nemusí.

RGB model můžeme taktéž vyjádřit 32 bity, kde prvních 24 bitů reprezentuje základní barvy a posledních osm bitů reprezentuje alfa kanál, pomocí kterého je možné přidat barvě průhlednost v intervalu od 0 – 255, kde 0 je pixel viditelný a 255 je zcela neviditelný.

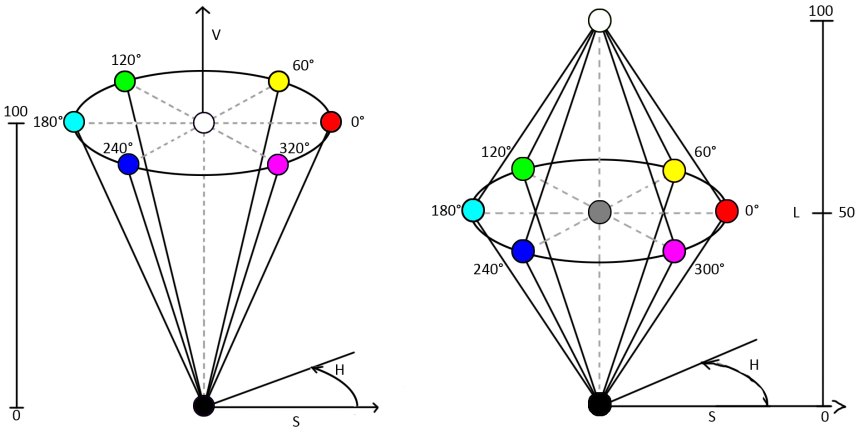
Opakem RGB je barevný model CMY, kde odpovídá sčítání hodnot v CMY subtraktivnímu skládání barev. Tedy vektor $\vec{v}\{0;0;0\}$ reprezentuje barvu bílou a v protilehlém rohu barvu černou $\vec{v}\{255;255;255\}$ a zbytek barevných odstínů se skládá z barvy tyrkysové - C, fialové - M a žluté - Y. Tento barevný model se hlavně používá v tiskárnách, kde je jako bílý podklad papír, a proto není nutno bílou barvu míchat k jejímu dosažení.

Převod mezi RGB a CMY je velmi jednoduchý. Stačí si jen uvědomit, že prostor je inverzní. Mějme tedy vektor v RGB, vyjádřený jako tříprvkovou matici $\vec{v_1}\{R;G;B\}$, z které lze vektor $\vec{v_2}\{C;M;Y\}$ určit následujícím odčítáním vektorů.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 255 \\ 255 \\ 255 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

2.1.2 HSV(HSB) a HSL

Přednost následujících modelů od modelu RGB nebo CMY je v tom, že se snaží popsat barvu na základě vnímaní lidským okem a snížit celkový počet barevných odstínů v modelu. V těchto modelech se barvy navzájem nemíchají, naopak jsou popsány přirozeným způsobem pro člověka. Lidské oko je nejvíce citlivé na intenzitu světla a tyto modely v sobě přímo tuto vlastnost obsahují. Další vlastností je barevný tón, který přesně odpovídá viditelnému spektru lidského oka a je seřazen podle vlnových délek od nejvyšší po nejnižší. Jako poslední je sytost barvy, což představuje intenzitu barvy.



Obrázek 2: HSV barevný model , HSL barevný model

Geometrickou reprezentací HSV je jehlan, jehož výškou vyjadřujeme intenzitu světla v rozmezí 0 až 100%, kde rostoucí procentuální hodnota určuje stále světlejší barvy. Obal jehlanu má hodnotu S = 100, kde rostoucí hodnota reprezentuje stále sytější barvy. Vzdáleností bodu od středu tedy rozumíme Jas (S), který určuje intenzitu sytosti barvy. Barevný tón nakonec znázorňuje úhel v rozsahu od 0 - 360 , na jehož počátku se nachází barva červená.

Počet odstínů v tomto modelu je o poznání nižší než v RGB. Počet všech kombinací třech prvků nám dává hodnotu 3 600 000, což je pro zpracování lidským okem rozhodně lepší. HSV má však dva hlavní nedostatky. Prvním je složitě určitelný přechod mezi černou a bílou a dalším je změna barevného tónu, která není zcela plynulá.

Tyto nedostatky řeší model HSL, který se stejně jako HSV skládá ze třech základních vlastností. Barevného tónu (Hue), sytost (Saturation) a intenzity světa (Lightness). Hlavním rozdílem je způsob, kterým je model reprezentován. Na rozdíl od předchozího HSV je složen ze dvou stejně velkých jehlanů, které jsou vzájemně spojeny podstavami. Tato vlastnost nám dovoluje plynulý přechod mezi bílou a černou a plynulý přechod v barevném tónu. Zbylé vlastnosti popisuje stejně jako model HSV.

Pro nedostatky modelu HSV se jím už dále zabývat nebudeme a ukážeme si pouze převod RGB do HSL. Prvním krokem je normalizace složek RGB.

$$r = \frac{R}{255}, g = \frac{G}{255}, b = \frac{B}{255}$$

Dalším krokem je nutné zjistit maximální a minimální hodnotu složek v RGB vektoru. Máme-li tyto informace splněny, tak je postup dán následujícími vztahy.

$$L = \frac{1}{2} * (min + max)$$

$$S = \begin{cases} 0 & \text{jestliže } min = max \\ \dfrac{max - min}{max + min} & \text{jestliže } L > 0 \text{ a zároveň } L \leq \dfrac{1}{2} \\ \dfrac{max - min}{2 - (max + min)} & \text{jestliže } L > \dfrac{1}{2} \end{cases} \tag{1}$$

$$H = \begin{cases} 0 & \text{jestliže } min = max \\ 60^\circ \cdot \dfrac{g - b}{max - min} + 0^\circ & \text{jestliže } max = r \text{ a } b \leq g \\ 60^\circ \cdot \dfrac{g - b}{max - min} + 360^\circ & \text{jestliže } max = r \text{ a } g < b \\ 60^\circ \cdot \dfrac{b - r}{max - min} + 120^\circ & \text{jestliže } max = g \\ 60^\circ \cdot \dfrac{g - b}{max - min} + 240^\circ & \text{jestliže } max = b \end{cases} \tag{2}$$

2.2 Barvy a jejich odstíny

Základním rysem obrázku jsou bezpochybně jeho barvy [5], a proto je nutné si pevně stanovit, které barvy chceme uživatelům systému nabídnout k vyhledávání. Každý tvůrce by byl rád, za co nejširší škálu barev ale v mé vlastní implementaci se později dozvíte, že to není zrovna rozumné řešení. Na tvorbě palety se často podílí několik lidí, právě pro jejich specifické vnímání barev a odstínů.

Při práci s RGB modelem je předem jisté, že určité barvy v obrázku nebudou, z důvodu obrovského množství barev, přiřazeny do námi vytvořené palety. V HSL modelu nastává podobný problém při jeho převodu z RGB, kde může dojít k překrývání určitých odstínů. Řešením tedy může být kombinace těchto dvou modelů.

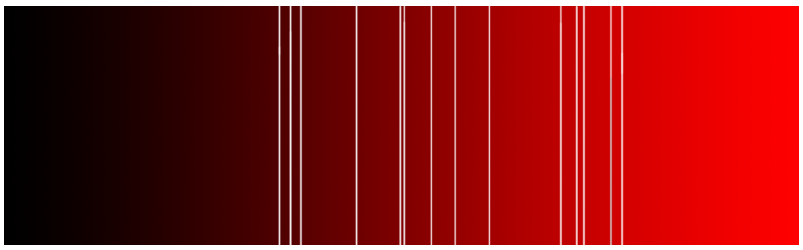
Poměrně známé základní barevné schéma pro případnou budoucí paletu je vyobrazeno v následující tabulce. Názvy nemusí být přesné, protože každý jedinec může mít určitý název barvy spojen s barvou jinou.

Color	R G B	H S L
Black	0 ; 0 ; 0	0 ; 0 ; 0
Sea green	0 ; 182 ; 0	120 ; 100 ; 36
Light green	0 ; 255 ; 170	160 ; 100 ; 50
Olive green	36 ; 73 ; 0	90 ; 100 ; 14
Aqua	36 ; 146 ; 170	191 ; 65 ; 40
Bright green	36 ; 255 ; 0	112 ; 100 ; 50
Blue	73 ; 36 ; 170	257 ; 65 ; 40
Green	73 ; 146 ; 0	90 ; 100 ; 36
Turquoise	73 ; 219 ; 170	160 ; 67 ; 57
Brown	109 ; 36 ; 0	20 ; 100 ; 21
Blue gray	109 ; 109 ; 170	240 ; 26 ; 55
Lime	109 ; 219 ; 0	90 ; 100 ; 43
Lavender	146 ; 0 ; 170	292 ; 100 ; 33
Plum	146 ; 109 ; 0	45 ; 100 ; 29
Teal	146 ; 182 ; 170	160 ; 20 ; 64
Dark red	182 ; 0 ; 0	0 ; 100 ; 36
Magenta	182 ; 73 ; 170	307 ; 43 ; 50
Yellow green	182 ; 182 ; 0	60 ; 100 ; 36
Flouro green	182 ; 255 ; 170	112 ; 100 ; 83
Red	219 ; 0 ; 0	0 ; 100 ; 43
Rose	219 ; 146 ; 170	340 ; 50 ; 72
Yellow	219 ; 255 ; 0	68 ; 100 ; 50
Pink	255 ; 36 ; 170	323 ; 100 ; 57
Orange	255 ; 146 ; 0	34 ; 100 ; 50
White	255 ; 255 ; 255	0 ; 0 ; 100

Tabulka 1: Zobrazení široké škály barev [5]

V tabulce 1 můžeme vidět poměrně velké množství barev, které jsme v běžném životě schopni jednoduše pojmenovat. Problém však spočívá v určení hranice mezi přechodem jednotlivých barev. Tato situace je velmi těžce řešitelná, právě pro rozdílnost vnímání barev jednotlivých lidí a také pro rozdílné zobrazovací zařízení. Můžeme s jistotou konstatovat, že každý jedinec má jinak nastavené jasové podsvícení monitoru, čímž dochází k výraznému zkreslení vnímání barvy. Pro dokázání tohoto jevu, byl vytvořen obrázek, kde černá barva plynule přechází do barvy červené. Lidé zde ve vlastních podmínkách

vyznačovali hranici mezi červenou a tmavě červenou barvou.



Obrázek 3: Demonstrační obrázek určující hranici mezi červenou a tmavě červnou

Bílé vertikální linky na obrázku 3 znázorňují hranice mezi dvěma odstíny červené, které vyplnili jednotliví lidé. Můžeme konstatovat, že rozdíl mezi nejvzdálenějšími hranicemi je natolik velký, že lze jen těžko mluvit o nějakém průměru, se kterým bychom mohli pracovat. Je proto nezbytné zredukovat počet barev v paletě a problém řešit jiným způsobem, který si představíme v praktické části.

2.3 Přiřazování odstínů do palety barev

V okamžiku, kdy je paleta určitým způsobem navrhnuta, je nutno promyslet, jak budeme přiřazovat barvy z obrázku k barvám v naší paletě.

Každý obrázek je třeba procházet určitým předepsaným algoritmem. Není však nutné procházet pixel po pixelu a poté načíst barvu z obrázku v RGB. Můžeme si předem převést obrázek do HSV modelu, načítat barvu z něj a následně ji přiřazovat do naší palety barev.

RGB nebo HSL jsou třídimenzionální prostory, a proto můžeme barvu označit jako prostorový tří prvkový vektor $\{r; g; b\}$ nebo $\{h; s; l\}$. Pro přiřazení barvy z obrázku do palety můžeme použít výpočet vektorové vzdáleností mezi dvěma vektory. V následujícím textu budeme popisovat 4 metody výpočtu těchto vzdáleností a také vytváření podprostorů jednotlivých barev v barevném modelu, z něhož bude barva z obrázku vyhledávaná.

2.3.1 Vektorová vzdálenost

Pro výpočet vzdálenosti dvou vekterou můžeme použít následující výpočty. Každá metoda výpočtu vzdáleností nám však dodá různé výsledky, proto není možné tyto metody mezi sebou kombinovat.

Máme tedy vektor $\vec{a} = \{2, 3, 4\}$ a vektor $\vec{b} = \{4, 3, 2\}$, který dosadíme do následujících metod. Vektor $\vec{a}$ představuje barvu načtenou z obrázku a vektor $\vec{b}$ je barvou z naší palety. Použitím jednoho z níže uvedených postupů vypočteme všechny vzdálenosti mezi barvou z obrázku a barvami z palety. Nejmenší nalezená vzdálenost určuje, o jakou barvu jde [14].

Euklidovská vzdálenost:

$$vzdlenost(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} = \sqrt{(2-4)^2 + (3-3)^2 + (4-2)^2} = 2.82 \quad (3)$$

Euklidovská vzdálenost čtverce:

$$vzdlenost(a,b) = \sum_{i=1}^{n}(a_i - b_i)^2 = (2-4)^2 + (3-3)^2 + (4-2)^2 = 8 \quad (4)$$

Manhattan:

$$vzdlenost(a,b) = \sum_{i=1}^{n}|a_i - b_i| = |2-4| + |3-3| + |4-2| = 4 \quad (5)$$

Canberra:

$$vzdlenost(a,b) = \sum_{i=1}^{n}|a_i - b_i| = |2-4| + |3-3| + |4-2| = 4 \quad (6)$$

Představené metody vektorové vzdálenosti jsou velice časově i výpočetně rychlé, avšak poměrně nepřesné. RGB model a i ostatní barevné modely mají totiž nelineárně rozložené barvy a přiřazování barev na základě vzdáleností v prostoru může přinést velkou chybovost. Z tohoto důvodu si představíme ještě jednu metodu, která řeší tento problém. Je velmi přesná ale za cenu větší časové náročnosti.

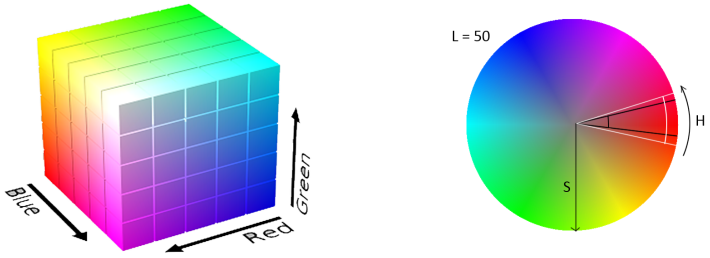
2.3.2 Vytváření barevných oblastí

Právě chybovost předchozích vektorových přiřazování byla určitou inspirací pro vytvoření této metody. Není totiž předem jasné, kde se barva v prostoru RGB nebo HSL nachází a jaký má rozsah. Z tohoto důvodu je nezbytné vytvořit oblast, ve které se vyskytují odstíny barev patřící jen jedné barvě z palety. Následně jen stačí zkontrolovat, zda se barva z obrázku nachází v prostoru dané barvy z palety.

Voxel (Volumetric element), neboli objektový prvek, je tzv. prostorový pixel umožňující uchovávat více odstínů jedné barvu v jednom elementu, tedy Voxelu. Ten si představme jako podmnožinu odstínů jedné barvy z celkové škály odstínů v RGB modelu. V praxi to znamená, že velikost Voxelu by měla být taková, aby všechny její vnitřní barevné odstíny patřily do odstínu jedné základní barvy z palety. Dále stačí jen porovnat, zda leží barvy z obrázku uvnitř Voxelu [14].

Problémem však je zajištění velikosti Voxelu ve všech třech osách tak, aby odstíny barev náležely jen k jedné barvě. Tento proces je velmi časově náročný, z důvodu nutných experimentů s lidmi, kterými docílíme co nejpřesnějších barevných prostorů. Nelze se však spokojit jen s jedním Voxelem barvy. V RGB modelu jsou barvy nelineárně rozloženy, což způsobuje, že množina Voxelů pro jednu barvu bude bezesporu větší. Pro představu počet Voxelů zelené barvy se pohybuje ve stovkách.

Stejně může vše fungovat v HSL prostoru, jen s rozdílem toho, že nebudeme pracovat s Voxelem ale jinou datovou strukturou. Místo využívání třech os bude třeba pracovat s H – úhlem, S – poloměrem a L – výškou. Pro inspiraci si představíme obrázek , který je vygenerovaný z programu na CD v příloze práce.



Obrázek 4: Rozložení prostoru na podprostory obsahující pouze jednu barvu

Na levé straně obrázku 4 jsou znázorněny Voxely v RGB prostoru a jejich rozmístění. Obrázek je pouze demonstrační, v praxi nebudou Voxely tak pravidelně rozložené.

Pravá část obrázeku 4 zobrazuje model HSL při intenzitě světla L = 50, kde je viditelný výřez odstínů červené barvy.

Vytvoření prostorů, v nichž se nachází odstíny jedné barvy, je u HSL i RGB velmi náročné, nicméně docílíme tím velmi kvalitních výsledku.

2.4 Segmentace obrazu

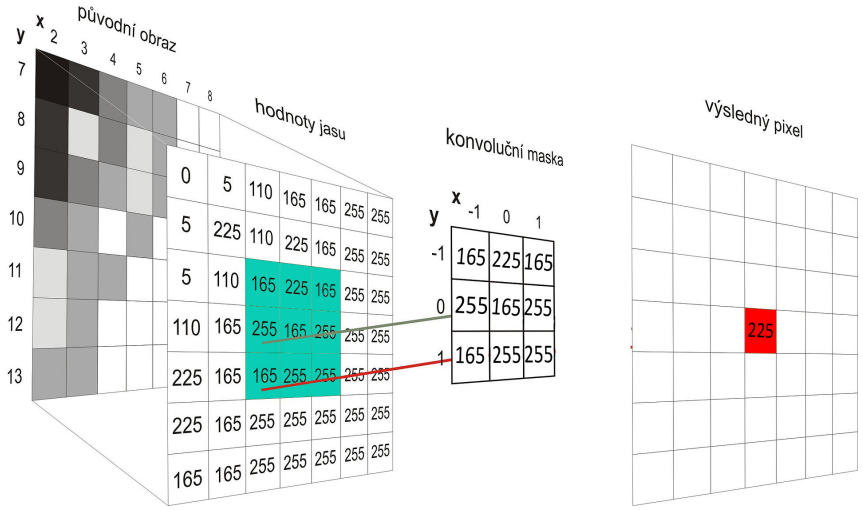
Dosud jsme si jen představovali způsoby vyhledávání barev v obrázku. Je však možné vyhledávat i jiné rysy v obrázku, což mohou být třeba geometrické tvary. V tomto případě je nutné obrázek rozdělit do logických celků tak, že každá oblast bude mít jednu společnou vlastnost (jas, barvu, texturu, atd..), z nichž si zvolíme jednu nebo kombinaci více vlastností. Tento proces se nazývá segmentace obrázku [3, 4, 6].

Jakmile se podaří rozdělit obrázek do oblastí se stejnou vlastností, můžeme vylepšit barevné vyhledávání přidáním analýzy místa, kde se daný objekt nachází. Je zřejmé, že mnohem větší prioritu bude mít objekt ve středu obrázku, než na jeho okraji, případně schovaný v jeho rohu.

I zde však mohou nastat komplikace. Jednou z nich může být nejednoznačnost obrazu nebo šum nacházející se v něm. Z toho důvodu provádíme na samém začátku segmentace odstranění nežádoucího šumu. Jednou z vhodných metod je použití **nelineárního mediánového filtru**.

Tento filtr je pro proces vyčištění obrázku nejvhodnější. Nijak nám nezkreslí obrázek a přitom nás poměrně dobře zbaví šumu. Filtr přiřazuje pixelu ve výstupním obraze

medián z okolí daného pixelu. Tento proces se nazývá konvoluce, což je matematická operace, která kombinuje dva obrazy tak, aby vznikl obraz třetí. První obraz je vstupní obrázek, druhý je konvoluční maska (podobraz vstupního obrázku) a třetí je výsledkem konvoluce. Velikost konvoluční masky může být jakákoliv, ovšem nejčastěji se užívá velikost strany 3 nebo 5.



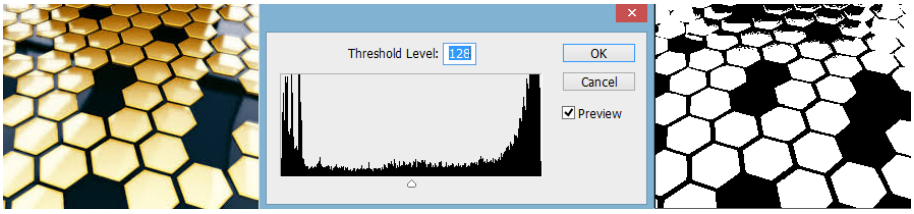
Obrázek 5: Mediánový filtr

Střed konvoluční masky je postupně kladen na všechny pixely ve vstupním obraze. Položímeli filtr na pozici vstupního obrázku x, y tak hodnoty okolních pixelů se poté seřadí vzestupně, nebo sestupně, dle jasových hodnot RGB. V případě, že počet hodnot je lichý, vezme se mediánový prvek ležící uprostřed seřazené řady. Pokud však bude sudý, mediánem bude aritmetický průměr dvou středových hodnot. Tato hodnota bude následně uložena do výstupního obrazu na pozici, kde se nachází střed filtru na vstupním obrazu.

2.4.1 Prahování

Je nejstarší a nejjednodušší metodou segmentace. Je časově i výpočetně nenáročná ale výsledky nedosahují požadované přesnosti, jak tomu bude u dalších metod. Prahování je vlastně transformace vstupního obrazu na nejčastěji binární (dvoubarevnou) formu obrazu. Je také možné užít metod, kdy je pro obraz vytvořen vyšší počet prahů než pouze dva. Výstupní obraz pak bude rozdělen do tolika odstínu šedí, kolik je vytvořených prahů pro obraz.

Prahování se nejčastěji užívá v jednodušších obrazech, kde se od sebe oddělují objekty a jednobarevné pozadí [1]. O každém objektu v obraze můžeme říct, že má jasně definovanou odrazivost nebo pohltivost svého povrchu. Pokud se tyto dvě vlastnosti příliš nemění, můžeme využít hodnoty jasu k oddělení objektu od pozadí, případně objektu od objektu.



Obrázek 6: Výsledek prahování

V první řadě se vstupní obraz převede pouze na obraz v odstínech šedí. Z takto nově vytvořeného obrazu dále vytvoříme **Histogram**, což je grafické znázornění popisující četnost jednotlivých odstínů šedí. Tato četnost je vyjádřena jako velikost sloupce v rozsahu $0 - 255$. Z takto vytvořeného histogramu je dále možné určit hodnotu, popřípadě interval prahu. Hranice prahu se nejčastěji urči jako lokální minimum mezi dvěma lokálními maximy nebo jako polovina vzdálenosti mezi dvěma lokálními maximy.

$$g(x,y) = \begin{cases} 1 & f_(x,y) \in A_1 \\ 2 & f_(x,y) \in A_2 \\ & . \\ n & f_(x,y) \in A_n \\ 0 & jinak \end{cases} \tag{7}$$

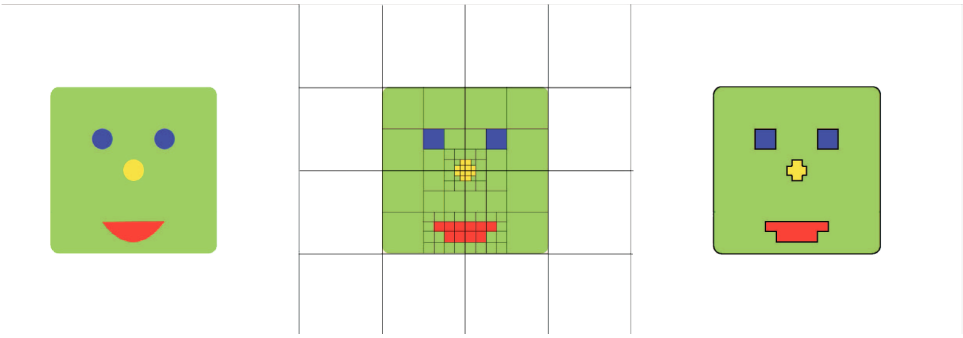
Kde funkce g (x,y) je vstupní hodnota obrazu, f (x,y) je jasová hodnota vstupního obrazu a A_x představuje hranici (interval) prahu.

V případě, že máme obraz s velkým množstvím jasových úrovní, nebude možné dobře určit prahy. Lze obraz rozdělit na podobrazy, kde každý z nich prahujeme lokálním prahem. Pokud ani v takto vytvořeném podobraze není možné určit prah, získáme jej interpolací sousedních prahů.

2.4.2 Segmentace založená na hledání oblastí

Stejně tak jako u předcházející metody prahování je nyní nutné definovat kritérium homogenity oblasti, které bude určovat, jak se objekty od sebe oddělují. Tyto kritéria mohou být založena na vlastnostech typu barvy, jasové složky, vzdálenosti barev v prostoru RGB aj.

Metoda vznikala ze dvou starších metod, a to **Region merging** a **Region splitting** [1]. Zde je obraz postupně dělen na pravidelné regiony a následně sjedoncení sousedících regionu za předpokladu splnění podmínky homogenity. Tento proces končí v okamžiku, kdy je obraz rozdělen do regionů, které jsou homogenní, a žádný sousední region již nelze spojit.



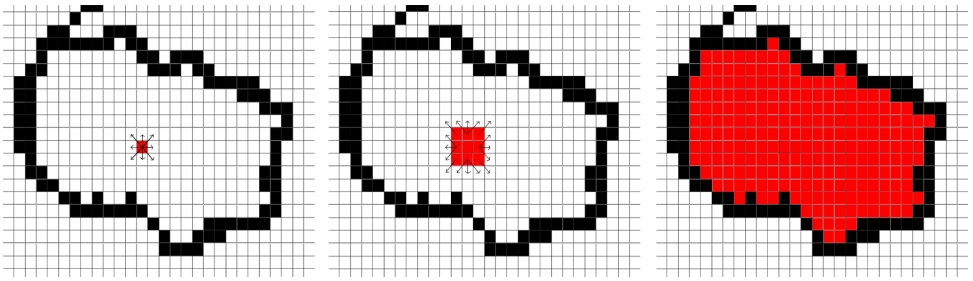
Obrázek 7: Původní obrázek, rozdělení na oblasti a následné sloučení oblastí (zleva)

U tohoto druhu segmentace je nutné definovat datovou strukturu, která se bude používat k uchovávání a správě nově vytvořených oblastí. Pro tuto segmentaci se užívá datová struktůra zvaná **Quad tree**, což je stromová struktura, kde každý vrchol stromu má čtyři větve. Výjimkou jsou listy, kde strom se dále nedělí.

2.4.3 Semínkový algoritmus

Tato metoda je společně s prahováním časově i výpočetně nejsnadnější metodou z této kapitoly [15]. Tento algoritmus opět potřebuje jasně stanovit podmínku homogenity, která je podmínkou k tomu, aby se algoritmus nezastavil před neúplnou detekci objektu v obrazu.

Jádrem metody je náhodný výběr místa (pixelu) v obraze a rekurzivní pohyb směrem od něj k okrajům obrazu. Způsob postupu od vybraného pixelu je zcela individuální. Je možné se pohybovat čtyřmi nebo osmi směry. Každá varianta má své klady i zápory. Budeme-li se šířit čtyřmi směry, nemusíme se v objektu dostat do všech částí, čímž může dojít k rozdělení objektu v obrázku na další dílčí objekty, což je nežádoucí. Pokud se však budeme pohybovat osmi směry, můžeme se dostat přes roh do jiného objektu, který má stejnou homogenitu, což je nežádoucí taktéž.



Obrázek 8: Postup semínkového algoritmu

Na obrázku 8 lze vidět, jak by se tento algoritmus mohl pohybovat. Je tedy nutné uchovávat binární pole o stejné velikosti jako je obraz, který bude uchovávat pozice, kde se algoritmus již nacházel, aby nedošlo k nekonečné rekurzi a výpočet mohl být ukončen.

Lze zde i jednoduše zjistit, zda se aktuální posuzována pixel nachází na hranice objektu nebo v něm. Pokud se z daného místa rozplyne do všech vysílaných směrů, je zřejmé, že leží uvnitř objektu. V opačném případě bude ležet na hranici.

2.4.4 Detekce hran

Dalším složitým a dosud nevyřešeným problémem je detekce hran [1, 4] v obrazech pořízených z reálných scén. Hranou rozumíme takovou část obrázku, kde se prudce mění
jasová hodnota mezi vzájemně sousedícími pixely. Hrana je tedy v podstatě vlastnost obrazového bodu započteného jako funkci obrazu v okolí tohoto bodu. Je reprezentovaná velikostí a směrem. Celý proces detekce hran lze rozdělit do třech základních úkolů, jimiž jsou filtrování, diferenciace a detekce. Filtrováním odstraníme nežádoucí vlastnosti obrazu, které vznikly při pořízení, což je šum nebo rozmazání. Diferenciací zvýrazníme oblasti, kde je intenzita obrazu významná. Jako poslední jsou detekovány a lokalizovány body, kde je významná změna intenzity jasu. Metody detekce hran lze rozdělit do dvou základních skupin. Metody využívající první derivace a druhé derivace. V praxi se pak využívá konvoluce obrazu s vhodně zvolenou maskou pro aproximaci první nebo druhé derivace.

Při použití metody první derivace je výsledkem gradient porovnávaný s prahem, který určuje, zda se jedná o hranu či nikoliv. První derivaci diskrétního obrazu získáme jako rozdíl mez okolních pixelů.

$$\Delta_x f(x,y) \approx f(x+1,y) - f(x,y)$$

$$\Delta_y f(x,y) \approx f(x,y+1) - f(x,y)$$

K výpočtu lze také přistupovat jako ke konvolučnímu filtrování, kde se jednotlivé hranové detektory liší jádrem konvolučního filtru. Ten nám určí, které body se pro výpočet gradientu použijí a jaké budou mít váhy. Příkladem mohou být následující konvoluční masky.

$$h_1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} h_2 = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} h_3 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} h_4 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Využitím druhé derivace je výpočet průchodu funkce nulou. V místě, kde je první derivace maximální, je druhá derivace rovná 0. Pokud můžeme v diskétním obraze aproximovat první derivaci jako rozdíl hodnot jasu dvou sousedních bodů, výsledná druhá derivace bude určena rozdílem těchto dvou sousedních rozdílů. Vztah pro oba směry bude vypadat následovně.

směr x :

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \big(f(x+1,y) - f(x,y)\big) - \big(f(x,y) - f(x-1,y)\big)$$

směr y :

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \big(f(x+1,y) - f(x,y)\big) - \big(f(x,y) - f(x-1,y)\big)$$

Nejznámější operátor druhé derivace je **Laplaceův operátor** Δ^2 pro který platí následující vztah.

$$\Delta^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial^2 x} + \frac{\partial^2 f(x,y)}{\partial^2 y}$$

Konvoluční jádra pro Laplaceův operátor operátor mohou být následující. Existuje však spoustu jiných alternativních masek.

$$h_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} h_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} h_3 = \begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix}$$

Výsledná derivace se na základě výše uvedeného vztahu pro Laplaceův operátor pro masku h_1 v diskrétním obraze se vypočítá následujícím vztahem:

$$\Delta^2 f(x,y) = f(x+1,y) + f(x-1.y) + f(x,y+1) + f(x,y-1) - 4 \cdot f(x,y)$$

Nevýhodou druhé derivace je její vysoká citlivost na šum, který je tedy nutné odstranit. Další nevýhodou je, že dochází k přílišnému vyhlazení obrazu, čímž dojde ke ztrátě ostrých rohů. V neposlední řadě se vyskytuje také sklon k vytváření uzavřených smyček hran.

2.5 Klasifikace objektu

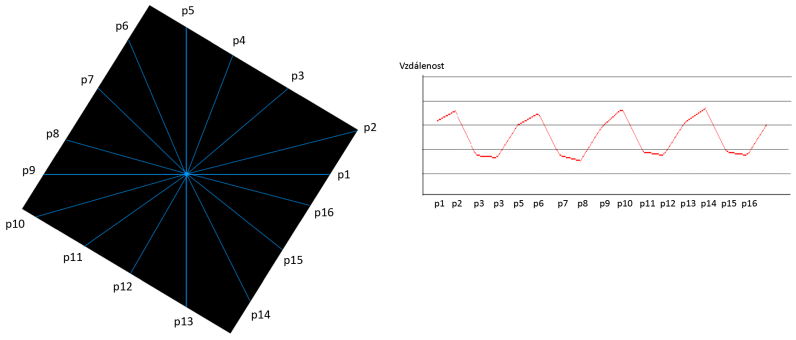
V předchozí kapitole jsme si popsali metody segmentace, které rozdělí obraz na jednotlivé logické celky. Nyní budeme tyto oblasti dále rozpoznávat do triviálních objektů, jako např. Kruh, čtverec, obdélník aj. Mimo samotné objekty lze rozpoznávat i jejich velikosti. Rozpoznávání všech těchto vlastností umožní dále popsané metody.

2.5.1 Funkce objektu

Tato metoda je sice výpočetně nenáročná ale obsahuje chyby v detekci malých objektů, případně selhává v detekci geometrických objektů, které mají atypické provedení. [8] Řešení těchto chyb si rozebereme ve vlastní implementaci.

Principem je vytvoření funkce, která určí počet lokálních minim a maxim daného objektu. Tyto lokální extrémy se musí dále ošetřit od nežádoucích doprovodných nízkých extrémů a následně funkci transportovat na obecnou velikost, která je pro všechny velikosti objektu stejná. Tímto zaručíme, že metoda bude invariantní vůči posunu i vůči rotaci.

Funkci lze vytvořit na základě paprsku jdoucího od středu objektu k jeho okraji. Nejprve paprsek detekuje průsečík hranice objektu a vypočítá vzdálenost od středu k lokalizovanému průsečíku objektu. Tuto hodnotu uloží, jako hodnotu funkce na pozici
vysílajícího paprsku. Počet paprsků je volitelný, avšak měl by být takový, aby umožňoval lokalizaci co největšího množství objektu. Příklad vytvořené funkce lze vidět na následujícím obrázku.



Obrázek 9: Ukázka funkce vypočtené ze čtverce

Na obrázku 9 lze vidět jak paprsky směřují od středu objektu k jeho okraji, vzdálenost každého paprsku je dále zobrazena na funkci zobrazující se v pravé části obrázku. Z takto vytvořené funkce můžeme říct že má 4 lokálni minima a 4 lokálni maxima. A pokud se lokální minima od sebe moc neliší co se týče vzdálenosti, stejně tak lokálni maxima, můžeme říct že v tomto případě se jedná o čtverec.

2.5.2 Příznakový vektor

Cílem této metody je matematická reprezentace objektu N dimenzionálním vektorem $\vec{x} = \{1, 2.., n\}$, kde každá dimenze zastupuje právě jeden příznak objektu. Příznakem můžeme rozumět vlastnost objektu, což může být barva, velikost, obvod, výška šířka aj. Příznaků může být nepřeberné množství, nicméně samotné nám k detekování přesného tvaru objektu nepomohou. Mnohem lepších výsledků docílíme pomocí tzv. momentů [5], kde momentový popis interpretuje jasovou funkci obrazu jako hustotu pravděpodobnosti dvojrozměrné veličiny. Máme obecný moment a centrální moment.

- **Obecný moment:**

$$m_{p,q} = \sum_x \sum_y x^p \cdot y^q f(x, y)$$

  Hodnoty $m_{p,q}$ obecného momentu nám určují charakteristiky daného momentu. Příkladem je moment $m_{0,0}$ určující hmotnost neboli plochu. Dále podíl prvního řádu $m_{0,1}$ nebo $m_{1,0}$ s momentem $m_{0,0}$ určující těžiště objektu.

$$x_t = \frac{m_{1,0}}{m_{0,0}}; y_t = \frac{m_{0,1}}{m_{0,0}}$$

  podíl momentu x_t a y_t nazýváme centroidy které dále využiváme k výpočtu centrálních momentů.

- **Centrální moment:**

$$\mu_{p,q} = \sum_x \sum_y (x - x_t)^p (y - y_t)^q f(x, y)$$

  Výpočet centrálního momentu je invariantní vůči posunu. Není tedy nutné znát přesnou polohu objektu, což umožní spolehlivě porovnávat další objekty navzájem.

Stačí tedy jen vypočítat příznaky, pomocí kterých budou objekty porovnávány, a ty nazveme F1 a F2

$$F_1 = \frac{obvod^2}{100 \cdot obsah}; F_2 = \frac{\mu_{min}}{\mu_{max}}$$

$$kde$$

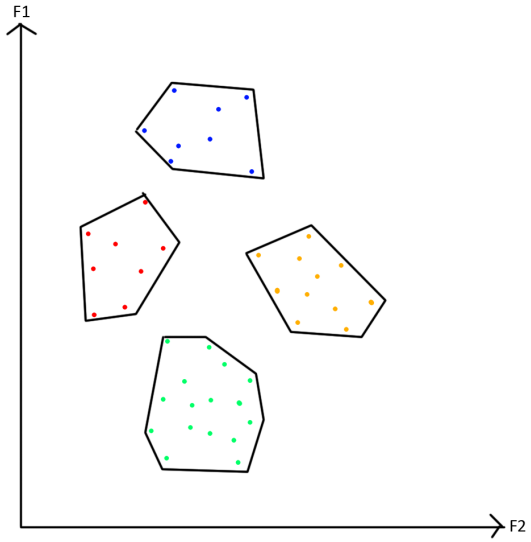
$$\begin{matrix} \mu_{max} \\ \mu_{min} \end{matrix} = \frac{1}{2}(\mu_{2,0} + \mu_{0,2}) \pm \frac{1}{2}\sqrt{4 \cdot \mu_{1,1} + (\mu_{2,0} - \mu_{0,2})^2}$$

$$obsah = \mu_{0,0}$$

$obvod =$ počet pixelů ležících na hraně objektu

Tímto výpočtem nám vznikne dvoudimenzionální příznakový vektor $\vec{v}\{F_1, F_2\}$ popisující objekt, jenž nám umožní mezi sebou porovnávat podobnosti ostatních objektů, nebo je nám umožní přiřazovat do námi vybraných tříd, což může být čtverec, obdélník aj. Vektory je tedy nutné porovnávat napříč prostorem, v kterém se nacházejí a zařadit je do správného podprostoru, v němž by se nacházet měli. Padne-li hledaný vektor do prostoru, kde se žádný hledaný objekt nenachází, můžeme tvar daného objektu ignorovat a dále porovnávat na základě jiných příznaků.

Pro správné vytvoření prostoru, kde se budou objekty nacházet, je nutné vytvořit trénovací množinu pro ty tvary objektu, které chceme vyhledávat. Tím docílíme vytvoření oblastí, kde by se měli objekty přibližně nacházet. Jako příklad rozdělení prostoru pro čtyři objekty máme následující obrázek. .



Obrázek 10: Možné rozdělení prostoru identifikující objekty

Na obrázku 10 jednotlivé barvy v prostoru reprezentují jeden objekt a je dále zřejmé, že čím větší bude množina pro objekty, tím budou přesnější výsledky.

3 Návrh a implementace vybraných nebo vlastních metod

Má vlastní implementace programu s sebou nesla od začátku mnoho problémů. Bylo nutné zkoumat různá řešení a provádět mnoho experimentů, abychom se dostali k vyhovujícím výsledkům, které budou vyhovovat co největšímu počtu lidí. Právě při vývoji programu bylo nutné zajistit co největší počet účastníků, kteří budou přispívat svými různorodými názory a specifickým vnímáním barev a jiných rysů. Dostávali jsme různá řešení při segmentaci obrázku nebo také výběru barevného prostoru, proto probíhaly po celou dobu implementace neustálé průzkumy a kontrolní testy.

Výsledné řešení obrázkového systému, založeného na různých rysech v obrázcích, nalezneme v následujícím textu, kde bude postupně popisován jeho podrobný vývoj.

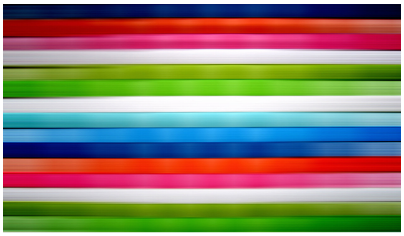
3.1 Anotovaná sada obrázků

Stavebním kamenem při vývoji systému je trénovací množina, neboli sada obrázků, s popisem rysů nacházejících se uvnitř. Ta nám velmi pomáhá při posuzování, zda daná metoda odpovídá realitě, nebo jestli neposkytuje zkreslené výsledky, což by znamenalo změnu přístupu k detekci rysů v obraze.

Sada obrázků by měla obsahovat všechny rysy, které chceme v systému vyhledávat a nejlépe ve větším množství. Různé sady anotovaných obrázků se dají běžně vyhledat a stáhnout, nicméně není zaručeno, že budou splňovat všechny naše požadavky. Řešením je tedy vytvořit naši vlastní sadu.

Jako první je tedy nutné vyhledat obrázky k testování a ty nabídnout účastníkům. Ti budou slovně hodnotit, co na daném obrázku vidí, pod čím by chtěli tento obrázek vyhledávat nebo čím je pro ně obrázek specifický. S rostoucím počtem lidí se nám bude samozřejmě zvyšovat počet různorodých výsledků, které bude třeba nějak sjednotit. Docházelo totiž k dohadům o tom, co jednotliví lidé vidí a jak to také chápou, a proto bylo nutné ignorovat určité popisy a pokračovat jen s nejčastějšími hodnoceními. V konečné fázi byl slovní popis od uživatelů převeden do binární formy pro všechny hledané rysy a uložen v souboru. Rys nacházející se v obrázku měl hodnotu 1 jinak 0.

Příkladem může být následující obrázek 20 s jeho výsledným binárním ohodnocením.



Obrázek 11: Ukázka obrázku z anotované sady

Není nutné používat trénovací množinu, avšak dosáhneme s ní lepších výsledků při vyhledávání s výhodou nižší náročnosti. Celá sada obrázků se nachází na CD v příloze této práce.

3.2 Výběr barev do palety

Výběr takových barev do palety, které by vyhovovaly všem uživatelům, je nekonečný proces. Právě z důvodu různorodosti uživatelských požadavků byla paleta několikrát měněna, až vznikla základní paleta o dvanácti barvách vyobrazena v tabulce 2.

Výsledná paleta vznikla na základě testu, kde účastníci slovně popisovali barvy z tabulky 1, které jim byly zobrazeny. Zobrazena sada barev obsahuje však i takové případy, pro které lze jen těžko vytvořit hranici, kde se od sebe jednotlivě oddělují. Tento problém je popsán jednoduchým testem v kapitole 2.2. Výsledky tohoto testu byly opět velmi různorodé. Z tohoto důvodu byly použity jen nejčastěji se vyskytující hodnoty. Barvy pojmenovány jen zřídka se z palety jednoduše vyřadily.

Color	R G B	H S L	
Černá	14 ;12 ; 15	51 ; 87 ; 3	
Červená	204 ; 30 ; 25	2 ; 78 ; 45	
Modrá	43 ; 95 ; 200	212 ; 96 ; 40	
Šedá	148 ; 150 ; 141	73 ; 4 ; 57	
Tyrkysová	61 ; 210 ; 211	180 ; 63 ;5340	
Zelená	73 ; 183 ; 51	110 ; 56 ; 46	
Hnědá	148 ; 86 ; 34	27 ; 63 ; 36	
Fialová	134 ; 47 ; 174	283 ; 58 ; 43	
Žlutá	227 ; 223 ; 58	59 ; 75 ; 56	
Růžová	231 ; 90 ; 171	326 ; 75 ; 63	
Oranžová	235 ; 129 ; 28	29 ; 84 ; 52	
Bílá	241 ; 240 ; 224	56 ; 38 ; 91	

Tabulka 2: Výsledná paleta barev

Vyobrazené RGB hodnoty vznikly z průzkumu odstínů daných barev, který se nachází v následující kapitole. Po vytvoření odstínových prostorů jednotlivých barev se RGB hodnoty nachází v jejich středech.

3.3 Zpracování barev z obrázku

Pro přiřazování True Color z obrázku do palety jsem pro její přesnost použil voxelovou metodu. Znamená to, že celý princip přiřazování bude založen na RGB modelu. Jediný problém tu představuje vytvoření oblasti, kde se nacházejí odstíny jedné barvy z palety. Výsledky jsou však v samém závěru nad očekávání dobré.

Pro vytvoření takové oblasti, jež bude popisovat jen jednu barvu, nebude stačit pouze jeden voxel. Bude nutné vytvořit takovou množinu, která nám tuto oblast co nejvhodněji ohraničí. Z tohoto důvodu byl vytvořen program, který nám bude nápomocen.

3.3.1 Experiment barevných odstínů z palety

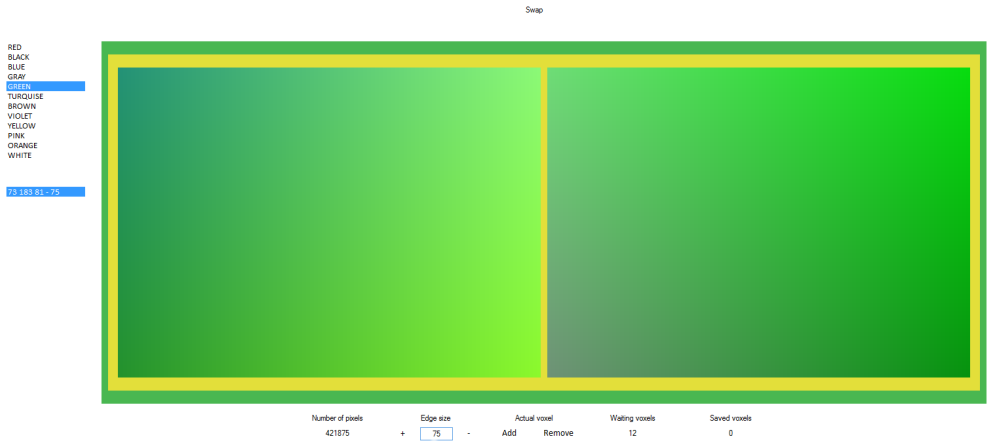
Základem program pro určování odstínu je seznamu barev zobrazený v tabulce 2. Účastník si tedy vždy vybral, pro kterou barvu bude vytvářen odstínový prostor a ta se mu zobrazila do náhledu programu.

Vybraná barva ze seznamu byla programem zobrazena pomocí voxelu o velikosti všech hran x. Nastavení hrany voxelu s sebou však neslo velká rizika. Při nastavení velkého rozsahu os voxel zabíral mnoho RGB odstínů a docházelo k problémům, kdy ve velkém prostoru barvy nesouhlasily s vybranou barvou. V případě nastavení malé velikosti rostl počet potvrzovaných voxelů do astronomických hodnot a docházelo ke hromadné chybovosti. Ta byla často důvodem nesoustředěnosti a únavy účastníka. Velikost hrany byla tedy ponechána na samotných respondentech, kteří ji mohli volně měnit na základě aktuální situace.

Pro představu, vždy po vygenerování základního voxelu účastník nastavil velikost hrany na takovou maximální velikost, kdy stále všechny odstíny ležící uvnitř voxelu odpovídaly vybrané barvě. Účastník jej potvrdil a došlo k uložení do výstupního souboru. Program následně vygeneroval nové voxely a proces se opakoval. V případě, že došlo k vygenerování takového voxelu, jehož vnitřní odstíny neodpovídaly vybrané barvě i při malých velikostech os, došlo k jeho zahození.

Nově vytvořené voxely se nachází vždy na pozicích obléhající právě potvrzený voxel, jejichž středy se nachází na jeho plášti. Tento proces se opakuje do doby, než se začnou objevovat odlišné odstíny barev od barvy hledané.

Ve chvíli, kdy již program nemá další voxel na zobrazení, uloží účastník své hodnoty a může pokračovat v barvě jiné. Uložené výsledky by měly mít takový formát, v němž bude možné pokračovat v systému na detekci rysů.



Obrázek 12: Náhled programu v experimentu pro průzkum odstínů barev

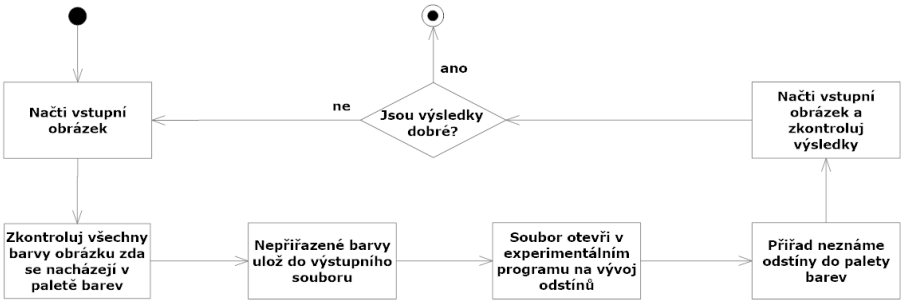
Pro dosažení co nejlepších výsledků byl v programu vytvořen specifický náhled, který pomáhal v rozhodování, zda vnitřní odstíny stále patří do posuzované barvy.

Na samém okraji náhledu je barva posuzovaná. Následující rámeček je tvořen všemi barvami z palety, které se v určitém časovém intervalu mezi sebou náhodně střídají. Tento prvek velmi ulehčoval rozhodování, protože se barvy na různém pozadí jeví lidskému oku vždy trošku jinak. Ve zbytku náhledu jsou vnitřní odstíny voxelů, na základě kterých byl daný případ potvrzen nebo zamítnut. Tento náhled zobrazoval odstíny barev ve voxelu od jedné z hran strany úhlopříčně přes celý voxel k jiné hraně. V poslední řadě si mohl uživatel zvolit jednu z osmi možností zobrazení vnitřních rozprostřených odstínů tak, aby byly výsledky co nejvěrohodnější.

Po dokončení experimentu s účastníky byly vytvořeny prostory v RGB, které odpovídaly jednotlivým barvám z palety. Výsledky byly vloženy do systému a zobrazeny každému účastníkovi ve formě obrázků z trénovací množiny.

Projevila se zde však nevýhoda RGB modelu, který byl popsán v kapitole 2.1.1. Výsledky nebyly u všech obrázků uspokojující, a proto bylo třeba přistoupit k druhému kroku průzkumu. Ten byl založen na vyhodnocování obrázku pixel po pixelu, kde byly nepřiřazené barvy do palety uloženy do výstupního souboru, jenž se následně načetl do experimentálního programu. Zde byl ke každé nedetekované barvě vytvořen voxel, který účastník individuálně přiřadil k jedné barvě v palety. U nejistých barev byl voxel automaticky zahozen. Při potvrzení voxelu se již žádné nové netvořily. Cílem tohoto snažení bylo pouze dosáhnout co nejlepších výsledků u daného obrázku.

Pro lepší pochopení je zobrazen následující diagram aktivit.



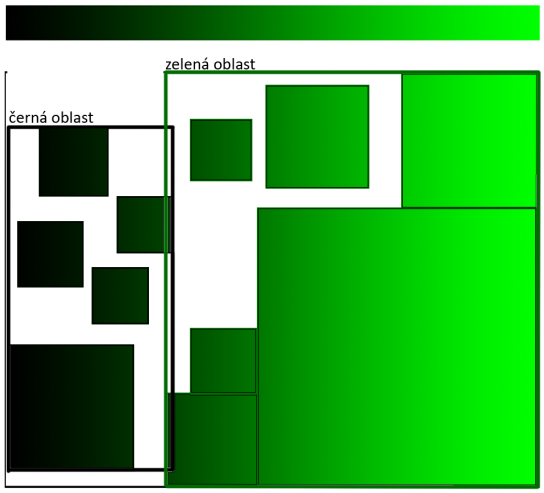
Obrázek 13: Diagram aktivit doprovodného experimentu

Po zavedení tohoto doprovodného experimentu lze říct, že bylo dosaženo očekávaných výsledku a je možné s nimi pracovat. Došlo však ještě k další malé komplikaci.
V průběhu dodatečného testu bylo zobrazováno velké množství odstínů barev, které ležely na hranicích mezi barvami z palety. Z tohoto důvodu občas docházelo k rozdílnému přiřazování a ve spojení s malou velikostí voxelu docházelo k jejich ohromnému nárůstu pro každou barvu. Pokud bychom tedy měli kontrolovat každý pixel, zda leží jeho RGB barva z obrázku uvnitř voxelu, algoritmus by se časově velmi prodloužil. Následující metody nám tedy tento problém vyřeší a významně algoritmus zrychlí.

3.3.2 Bounding box

Bounding box je (dále jen "BB") [13] libovolný kvádr v prostoru, reprezentující oblast, kde se nachází objekty se stejnou vlastností. Jeho velikost by měla být taková, aby uchovávala všechny své elementy a zároveň nebyla zbytečně větší. V našem případě bylo vytvořeno tolik BB, kolik máme barev v paletě a každý z nich bude uchovávat všechny voxely, reprezentující odstíny právě dané barvy.

Velikost BB lze zjistit u samotného načítání voxelu, kdy se hledá minimální a maximální hodnota na všech osách RGB. Po nalezení těchto hodnot vzniká na každé ose oblast, ve které se nachází všechny voxely jedné barvy z palety. Takto vytvořená oblast v prostoru RGB slouží k prvotní kontrole při přiřazování true color do palety. Po načtení pixelu z obrázku je v první řadě kladen dotaz na BB, zda leží barva uvnitř prostoru. Bohužel však není zaručené, že nalezení barvy v BB znamená zároveň její existenci uvnitř voxelu. V případě, že je barva odmítnuta směřuje pokus k další barvě z palety, a to až do jejího nalezení.



Obrázek 14: Bounding box

Na obrázku 14 lze vidět zjednodušený model dvou BB zobrazující 2D prostor kde v jednom se nachází odstíny barvy černé a v druhém odstíny barvy zelené.

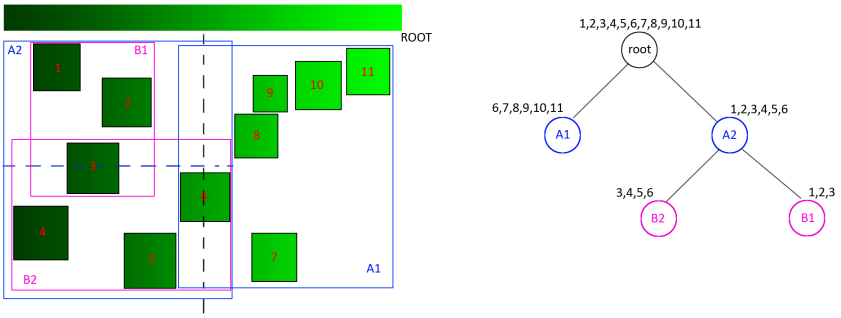
Touto jednoduchou implementací došlo k výraznému zrychlení detekce barev, kde maximální počet iterací na jeden obrazový pixel udává barva, která je reprezentovaná největším počtem voxelů. Stále je však třeba procházet všechny voxely v BB, proto je v další metodě rozebráno vyhledávání voxelů v prostoru, kterým opět dosáhneme rychlejšího nalezení barvy.

3.3.3 BSP strom

Binary space partitioning tree (dále jen „BSP“) [12] je objektový model, který byl vyvinut pro řešení správné viditelnosti navzájem se zakrývajících polygonů v počítačové grafice. Naše využití bude hlavně pro zrychlení hledání barvy v RGB.

BSP je úplný binární strom, kde každý uzel má právě dva syny, v jejichž uzlech se uchovávají informace o dělící nadrovině. Představme si RGB model jako hlavní uzel stromu (ROOT) obsahující množinu voxelů. Tuto množinu setřídíme podle kterékoliv ze tří os RGB a nalezneme voxel uprostřed setříděné posloupnosti. Střed nalezeného voxelu zvolíme jako bod na ose, podle které se třídilo, a na kterém bude RGB prostor rozdělen nadrovinou na dvě části. Voxely nacházející se nalevo od nadroviny se uloží v levém potomku uzlu a ty napravo v pravém potomku uzlu.

V takto nově vytvořených podprostorech tento postup znovu opakujeme, jen zvolíme jinou osu pro rozdělení prostoru a pokračujeme, dokud není splněná podmínka maximálního množství voxelů v uzlu. Takový uzel již bude listem stromu, který bude uchovávat výslednou množinu voxelu v podprostoru.



Obrázek 15: Rozložení prostoru: BSP tree

V levé části obrázku 15 vidíme postupné rozdělování prostoru až na prostory, v kterých se nachází minimální množství elementů. V pravé částí obrázku15 je znázorněný BSP strom uchovávající voxely.

Vyhledávání voxelů v RGB začíná v celém prostoru, který reprezentuje hlavní uzel stromu (Root), ve kterém je definovaná poloha nadroviny na určité ose. Stačí tedy napsat podmínku, do jaké části podprostoru patří hledaná barva podle polohy hledané barvy od definované nadroviny. Podle této podmínky se posuneme ve stromu o řád, kde tento postup opakujeme až do doby, kdy jsme v uzlu, který nám prostory dále nerozděluje. V každém posunu BSP stromu se zbavíme zkoumání téměř poloviny možných voxelů. Počet iterací hledání voxelů je tedy roven počtu úrovní stromu plus počet voxelů v posledním nalezeném podprostoru.

V našem případě není nutné tento algoritmus používat na celý prostor RGB. V kombinaci s předchozí metodou BB můžeme pro každý BB vytvořit vlastní BSP strom, a tak zrychlit vyhledávání jednotlivých barev. Výsledný algoritmus je již natolik rychlý, že není třeba dalších úprav pro zrychlení.

3.4 Segmentace

V této kapitole se budeme zabývat rozdělením obrázku do logických částí, což se nazývá segmentace. V samém začátku je důležité si obrázek předpracovat. Rozumíme tím kupříkladu odstranění šumu z obrázku, zaostření jeho hran aj. Následně už jen probíhá aplikace určité metody, která nám zajistí co nejlepší výsledky v podobě objektu se stejnou homogenitou v obrázku.

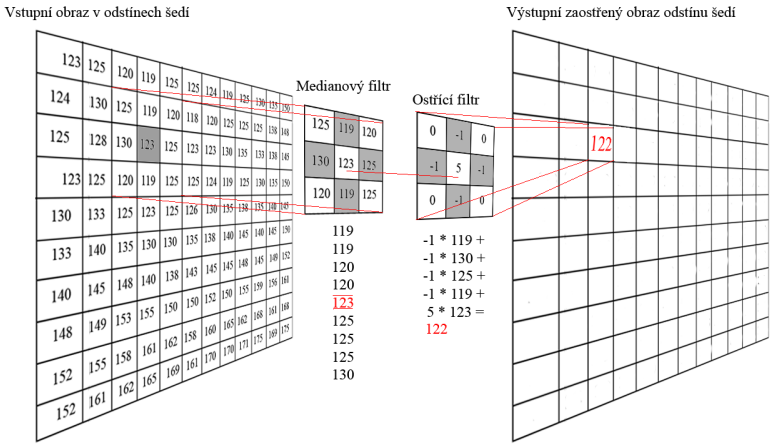
3.4.1 Předzpracování obrázku

Není nutností vždy provádět předpracování obrázku, avšak pokud k tomu dojde, je třeba vědět, že samotný proces se odvíjí od metody, která je použita a od druhu podmínky homogenity pro jednotlivé oblasti. V případě takové podmínky, kdy jednotlivé logické celky uchovávají jen jednu barvu, není třeba žádné úpravy obrázku. Pokud se však hledají vícebarevné logické oblasti, je předpracování nutné.

Obrázky s velkým rozlišením jsou často kvalitní (neobsahují mnoho šumu) a mají podlouhlé plynulé přechody mezi objekty. V takovém typu obrázku je tedy potřeba jen zaostřit hrany, v našem případě konvoluční maskou. Tato maska má však za následek zvýšení šumu, je tedy nutné vzniklý šum potlačit mediánovým filtrem. Kombinací těchto dvou operací vzniknou již vhodné výsledky pro následující semínkový algoritmus. U obrázků malých rozlišení, kde jsou přechody ostřejší a s výraznějším šumem, není vhodné používat konvoluční masku zaostřující hrany, a tak šum ještě zvýraznit. Pro tento typ případu je zvolen pouze mediánový filtr pro odstranění šumu.

$$\text{Konvoluční maska na ostření hran: } \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

V první řadě jsem je tedy vstupní obrázek převeden pouze do barev odstínů šedi. Ve všech složkách RGB se tedy nachází stejná hodnota, čímž máme vypočítanou svítivost pixelu. U velkých obrázků byl na každý pixel položený střed zaostřující konvoluční masky, kde středová hodnota masky byla určena mediánovým filtrem. Pro lepší znázornění této kombinace konvolučních masek využijeme obrázek, který poukazuje na výpočet hodnoty pixelu na jednom místě vstupního obrázku.



Obrázek 16: Kombinace mediánového a ostřícího filtru

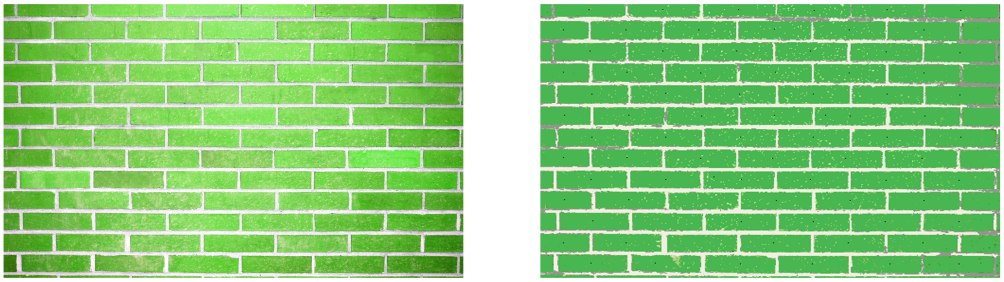
V případě, že je na vstupu obrázek s nízkým rozlišením, úprava obrázku probíhá obdobně, jen bez zaostřovacího filtru. Výsledný pixel na výstupním obrázku bude tedy vypočten pouze mediánovým filtrem.

3.4.2 Aplikace semínkového algoritmu

Pro detekci oblastí se stejnou homogenitou byl vybrán semínkový algoritmus,teoreticky popsán v kapitole 2.4.3. Výhodou této metody je její rychlost, poměrně snadná implementace a uspokojivé výsledky v detekci oblastí, a celkově obstál lépe než jiné metody segmentace.

Metoda je založena na rozplývání pixelů do okolí od zvoleného pixelu, za podmínky platné homogenity dané oblasti, kterou určí první náhodně zvolený pixel v obrázku. Nejdůležitější na této metodě je zvolit vhodnou podmínku homogenity, na které závisí detekování dané oblasti. V našem případě byla jako podmínka zvolena hodnota prahu. Tato hodnota byla pevně stanovená na hodnotu 12,5, jenž se při testování nejvíce osvědčila. Tato hodnota představuje rozpětí pro jednotlivou hodnotu barvy šedi nahoru i dolů. Tyto odstíny se nachází v RGB prostoru na diagonále mezi vrcholy $\{0, 0, 0\}$ a $\{255, 255, 255\}$. Díky stanovené hodnotě prahu můžeme konstatovat, že pokud se hledaná barva nenachází v určeném rozpětí, lze posuzované odstíny považovat jako hranici objektu. Semínkový algoritmus tedy posuzuje vzdálenost mezi náhodným pixelem a okolním pixelem, a rozšíří se v případě, že jejich vzdálenost spadá do rozpětí prahu. Pokud se nerozšíří, vznikne hranice objektu. V opačném případě se daný objekt zvětší. Tento proces probíhá až do okamžiku, kdy se nemá algoritmus kam rozšířit.

Na následujícím obrázku 17 lze vidět možný výsledek této metody segmentace.



Obrázek 17: Výsledek segmentace semínkovým algoritmem

Vlevo vidíme obrázek před segmentací a vpravo nalezené oblasti se stejnou homogenitou. Černé tečky zde znázorňují středy jednotlivých oblastí, jejichž počet určuje počet oblastí v obrázku se stejnou homogenitou.

3.5 Klasifikace objektů v obraze

Po nalezení částí obrázku předchozím semínkovým algoritmem můžeme přejít na další rys, spočívající v rozpoznání objektu, jenž reprezentuje část obrazu a je pro uživatele systému dále nabídnut. Nacházet můžeme tvary typu čtverec, obdélník, elipsa, kruh, trojúhelník a další mnohoúhelníky.

První otázkou je pro nás určení velikosti objektů, které má smysl v obraze vyhledávat. Může totiž dojít k nalezení natolik malých objektů, že budou pro uživatele bezpředmětné. Navíc by v takových případech mohl být lokalizován tvar, aniž by byl vidět. Proto byla minimální velikost objektu stanovena na pět desetin procenta obsahu obrázku. Dá se ovšem setkat i se situacemi, kdy je tato hodnota příliš malá nebo velká, proto ji nelze brát jako obecné pravidlo.

Pro identifikaci objektu byla zvolena metoda založená na vytváření funkce vzdálenosti od středu objektu k jeho okraji [8]. Tuto metodu lze začít řešit již při samotné segmentaci semínkovým algoritmem, který se dostane na všechny pozice pixelu v objektu. Právě toho lze využít pro vypočítání středu objektu. Jednoduše sečteme pozice pixelu x, y, kde se nacházel semínkový algoritmus, a tento součet následně vydělíme hodnotou počtu pixelů reprezentující zkoumanou oblast, čímž získáme střed objektu.

$$x_0 = \frac{\sum_{i=1}^{n} x_i}{n} \; ; \; y_0 = \frac{\sum_{i=1}^{n} y_i}{n} \tag{8}$$

Z takto vypočitaného středu jsou dále vysílány paprsky všemi směry s cílem nalézt okraj objektu. Využíváme k tomu boolovské pole obsahující pozice pixelů reprezentující okraj objektu vytvořeného ze segmentace. Každý analyzovaný pixel je srovnáván s hodnotami v boolovském poli, kde jednička představuje okraj a nula jen volný prostor. Na základě získaných hodnot vzdáleností vytvoříme funkci, podle které navzájem posuzujeme nalezené objekty.

Paprsky lze posílat od středu k okraji pomocí vztahu parametrické kružnice.

$$\begin{aligned} x &= x_0 + r \cdot cos\varphi \\ y &= y_0 + r \cdot sin\varphi \end{aligned} \tag{9}$$

Zde je x_0 a y_0 pozice středu kružnice neboli střed objektu. r je poloměr (proměnná hodnota) a $\varphi \in < 0; 2 \cdot \pi)$ je úhel, pod kterým paprsek směřuje. Pokud bychom měli počítat parametrickou kružnici pro každý paprsek v každém poloměru goniometrické funkce, algoritmus by se znatelně zpomalil. Z tohoto důvodu je vhodné tyto hodnoty pro každý paprsek předpočítat. Mějme například 16 paprsků označených p_0 až p_{15}. Pro každý paprsek se úhel vypočítá z následujícího vztahu.

$$q = \frac{2 \cdot \pi}{p} \tag{10}$$

kde p je počet paprsků výsledek. za pomocí tohoto propočtu dále už lze určit pod jakým uhlem budou paprsky vysílány.

		x	y
P_0	=	$\cos(q \cdot 0)$	$\sin(q \cdot 0)$
P_1	=	$\cos(q \cdot 1)$	$\sin(q \cdot 1)$
P_2	=	$\cos(q \cdot 2)$	$\sin(q \cdot 2)$
	.		
	.		
P_{15}	=	$\cos(q \cdot 15)$	$\sin(q \cdot 15)$

Máme-li předpočítány úhly pro jednotlivé paprsky, lze přistoupit ke hledání hranice objektu. V každé iteraci se navýší poloměr r v rovnici parametrické kružnice o jednotku, a tím se navýší vzdálenost paprsku od středu objektu. Z parametrické rovnice výsledky x, y udávají pozici paprsku v obraze, který se zkontroluje v pomocném poli, zda se jedná o hranu objektu či nikoli. Po nalezení takového poloměru, analyzující pozici reprezentující hranu, se vypočítá vzdálenost dle vztahu 3 Euklidovsko vzdáleností mezi středem objektu a pozicí, na které paprsek analyzoval hranici objektu. Tato vzdálenost se dále uloží jako hodnota funkce na pozici vysílaného paprsku. Funkce bude mít tedy takový počet hodnot, jaký je počet vysílaných paprsků.

Takto vytvořená funkce však stále podává zkreslené informace o objektu, protože pro stejný objekt, lišící se pouze ve velikosti, by tato metoda vytvořila různé hodnoty. Je tedy vhodné hodnoty funkce transportovat do jednotného měřítka. To lze zrealizovat metodou **Okénkové transformace**. Mějme tedy obecné hodnoty funkce, které přepočítáme do měřítka, kde maximální hodnota funkce bude například 100. V prvním kroku nalezneme maximální hodnotu vzdálenosti hrany od středu objektu vytvořené funkce, a z té vypočítáme parametr, jako podíl hodnoty měřítka s nalezeným maximem funkce. Tímto parametrem dále vynásobíme všechny hodnoty funkce, čímž vznikne funkce nová, která má své hodnoty již transportovány do jednotného měřítka. V každém případě bude mít

každá funkce minimálně jednu hodnotu o velikosti 100. Po této úpravě hodnot funkce máme zajištěnou invarianci vůči posunu i rotaci.

Nyní je pro tuto funkci potřeba naleznout její lokální extrémy, pomocí kterých se objekty budou přiřazovat do útvarů. Ty lze jednoduše naleznout, tak že pro každou hodnotu funkce budou kontrolovány její okolní hodnoty [9].

Lokální minimum: Funkce f má v bodě x_0 lokální minimum, existuje-li okolí bodu x_0 tak, že $\forall_x$ z tohoto okolí je $f(x) \geq f(x_0)$

Lokální maximum: Funkce f má v bodě x_0 lokální maximum, existuje-li okolí bodu x_0 tak, že $\forall_x$ z tohoto okolí je $f(x) \leq f(x_0)$

Podle počtu lokálních minim a maxim již lze odhadnout, o jaký typ objektu se jedná. Příkladem mohou být 4 lokální minima a 4 lokální maxima, o kterých můžeme říct, že by to mohl být čtverec. Pokud však nenalezneme ani jeden lokální extrém jedná se o kruh. Na místě je třeba zmínit, že lze najít situaci, kdy obsahují určité tvary stejný počet lokálních extrémů. Tvarově se však už neshodují s našimi hledanými objekty. Pro řešení tohoto problému existují dvě metody. Jednou z nich je určení intervalů, v nichž se lokální extrémy pro daný objekt pohybují. Je zřejmé, že u čtverce by měli mít všechny lokální minima přibližně stejnou hodnotu a stejně tak i lokální maxima. Tyto intervaly lze získat na základě pozorování a testování objektů, nebo si vytvořit trénovací množinu pro každý hledaný objekt, podle níž se budou porovnávat nalezené extrémy. Tato trénovací množina by měla obsahovat dostatečný počet případů pro jednotlivé objekty, jakými je lze v obraze detekovat, což je velice obtížné. Může totiž dojít k situaci, kdy hledáme objekty trojúhelníkového tvaru a narazíme díky nim na tisíce různých tvarů.

3.5.1 Řešení chybných klasifikací

Při počítání hodnot funkce obsahující vzdálenosti středů k okraji může docházet k chybám ve výpočtech. Existují totiž situace, kdy se jednotlivé objekty nachází uvnitř jiných objektů. Tento jev vyjadřuje špatně lokalizovaný střed objektu, který je nutno přepočítat. Řešení přepočtu lze provést na základě pixelů lokalizovaných paprsky. Vypočítá se průměrná pozice x a y všech lokalizovaných pixelů a porovná se prvotním středem. V případě, že se nově nalezená a původní pozice středu objektu od sebe neliší, výpočet se ukončí. V opačném případě se uloží pozice nového středu a znovu se provede vysílání paprsku od nového středu objektu k jeho okraji. Daný krok se opakuje až do fáze, kdy se střed nemění.

Další z problémů může nastat při posuzování malých objektů, kde je hustota vysílaných paprsků k okraji objektu příliš velká. To může mít za příčinu chybné lokalizování lokálních extrémů. Daný problém by však nenastal, kdyby byl obvod objektu popsán vektorově, nicméně rastrový obrázek nám tuto možnost neposkytuje, což by znamenalo jejich složité dopočítávání. Další z možností, jak se lze zbavit nežádoucích extrémů je aproximace, neboli přiblížení funkce. To lze zrealizovat například proložením funkce **Bézierovou křivkou** [11]. Ta vždy prochází prvním a posledním počítaným bodem funkce, a zbytek bodů, ležících mezi prvním a posledním, jsou aproximovány. Nejlepší výsledky

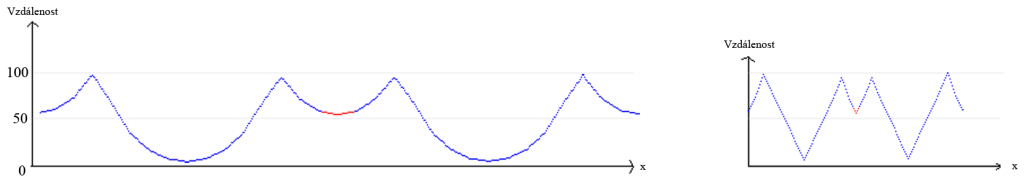
poskytovala **kubická Bézierova křivka**, která vždy procházela první a čtvrtou hodnotou funkce, dále čtvrtou až sedmou, sedmou až desátou atd. Výpočet bodů bézierový křivky lze uskutečnit pomocí následujícího vztahu.

$$C_{(t)} = \sum_{i=0}^{n} B_{i,n}(t) \cdot P_i$$

přičemž $B_{i,n}(t)$ je i-tý Bernsteinův polynom n-tého stupně:

$$B_{i,n}(t) = \binom{n}{k} t^i (1-t)^{n-i} \text{ kde } t \in < 0, 1 >$$

Dalším vylepšením průběhu již z aproximované funkce je projít funkci bod po bodu a zbavit ji nežádoucích hodnot malých skoků vzdáleností vzniklých při detekci malých objektů. Mějme tedy hodnotu funkce $f_{(x)}$, jejíž následující bod $f_{(x+1)}$ má hodnotu stejnou nebo minimálně lišící (+-1 jednotka vzdáleností). Takovou hodnotu odstraníme a pokračujeme dále na bod $f_{(x+2)}$. Zde probíhá stejná kontrola a případné odstranění hodnoty. Tímto způsobem lze funkci zbavit malých lokálních extrémů, které jsou nežádoucí. Upravenou funkci lze vidět na následujícím obrázku 18, kde po úpravě vzniky pouze ostré lokální extrémy.



Obrázek 18: Úprava průběhu funkce objektu typu obdélník

Červeně označená oblast v pravé částí obrázku 18 jsou hodnoty zaokrouhlené na celá čísla. V dané oblasti může při reálných hodnotách nastat až deset lokálních extrémů. V levé části může ovšem nastat pouze jeden, a to pouze ten, který je pro tuto metodu důležitý.

Za předpokladu, že budou prováděny tyto úpravy funkce, je možné nastavit i větší počet paprsků detekujících hranu objektu, což by umožnilo se dostat až do všech možných koutů objektu. Nežádoucí lokální extrémy budou takto následně vymazány.

3.6 Výpočet výsledných rysů

Shrnutím všech dosavadních procesů máme nyní rozložený obrázek na logické části, a o každém nalezeném objektu můžeme říct jakou má barevnost, pozici velikost apod. S danými informacemi můžeme dále libovolným způsobem pracovat a vytvářet různé vyhledávací rysy. Těch však může být nespočetně mnoho, proto si zde popíšeme jen ty nejzákladnější a nejvíc žádané u vyhledávání obrázků.

3.6.1 Triviální rysy

Pod triviálními rysy se skrývají běžně užívané obrazce, které nepotřebují žádnou složitou analýzu obrázku a jsou jednoduché k jejich získání. Právě pro jejich celkovou jednoduchost jsou v mnoha vyhledávacích systémech k nalezení.

Vůbec nejčastějším takovým rysem je název nebo určitý popis obrázku, udávající informace o tom, co se v něm vyskytuje. Jedná se o velice věrohodné vyhledávání, protože je tvořeno samotným uživatelem, který má nejlepší přehled o svých obrázcích. Stačí pouze nahrát řetězec do databáze a následně vyhledávat podřetězec v uložených názvech nebo popisech obrázku.

Další z řady základních rysů je například rozlišení obrázku. To může být uživatelům nabídnuto hned v mnoha variantách. Příkladem může být vyhledávání ve skupinách malých, středních nebo velkých obrázků, kde se jen určí interval počtu pixelů obrázku. Dále můžeme uživatele nechat, ať si sám zadá přesné rozlišení nebo interval. V neposlední řadě mu mohou být nabídnuta známá rozlišení z monitoru.

Můžeme také vyhledávat na základě polohy obrázku, která určuje způsob jeho vyfocení nebo vytvoření. Tento rys lze rozdělit do třech kategorií. Vertikální, horizontální nebo čtvercové, kde stačí jen porovnat výšku a šířku obrázku pro jeho vyhodnocení.

Jako doplněk může být nápomocna i výhoda zabudované GPS ve fotoaparátech, pomocí které lze obrázky vyhledávat i podle polohy, kde byly vyhotovené v rámci celého světa nebo času jejich uložení na server.

3.6.2 Barevné rysy

Vyhledávání dle barev je dalším hojně využívaným rysem. Můžeme jej nabídnout ve variantách pro vyhledávání pouze jedné nebo mnoha barev v obrázku. Podmínkou je zde určení váhy barev, která bude poskytovat informaci o množství barvy v obrázku.

V tabulce 2 je výčet barev, které jsou nabídnuty uživatelům k vyhledávání. V kapitole 3.3.1 jsou dále vytvořené odstíny jednotlivých barev získané na základě experimentu. V poslední řadě jsme v kapitole 3.4.2 semínkovým algoritmem detekovali objekty v obrázku, kde pro každý objekt je možno uchovávat jeho barevnost, velikost ale také pozici v obrázku. Všechny tyto informace lze zkombinovat tak, aby vznikly věrohodné vyhledávací informace o barvách.

V následujícím textu popíšeme výskyt barvy v obrázku, jenž je přiřazován do intervalu <0 - 10>, kde nula znamená nulový výskyt barvy, a deset je obrázek obsahující pouze jednu barvu.

Nejprve je nutné zjistit velikost objektu vůči celému obrázku, a to nejlépe procentuálně v intervalu 0 - 1. Dále vypočítat procentuální výskyt pro každou barvu v objektu, protože se nepředpokládá, že objekty v obraze budou obsahovat pouze jednu barvu. Už jen díky nadefinovaným podmínkám homogenity při segmentaci počítáme se situacemi, kdy jeden objekt obsahuje více než jednu barvu. Díky následujícímu vztahu vypočteme výskyt barvy v objektu.

$$C_x = \sum_{i=0}^{n} T_i \cdot (\frac{100}{O_i} \cdot C_b) \tag{11}$$

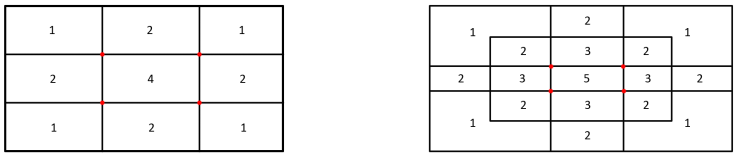
Kde výsledek C_x je hodnota pro jednotlivé barvy z palety, n je počet objektů v obraze, T_i vyjádřuje velikost jednotlivých objektu vůči celému obrazu vyjádřena v procentech v intervalu $< 0; 1 >$, O_i je obsah objektů nebo-li počet pixelu reprezentující objekt a C_b je počet pixelu v objektu dle počítané barvy C_x.

Dalším aspektem může být velikost objektu vůči obrázku. Co se týče vnímání barvy, tak objekt zabírající 50% obrázku bude mít jistě vyšší prioritu než dva objekty zabírající 25% obsahu obrázku té stejné barvy. K naplnění tohoto prvku se nadefinují procentuální intervaly, které vrátí hodnotu, pomocí níž pronásobíme výsledek vztahu 11.

$$Q_{(x)} = \begin{cases} 1.5 & O_i > 0.5 \text{ objekt je větší jak polovina obrázku} \\ 1.3 & O_i > 0.3 \text{ objek má velikost mezi 30 až 50 procenty obrázku} \\ 1.1 & O_i > 0.1 \text{ objek má velikost mezi 10 až 30 procenty obrázku} \\ 1 & jinak \end{cases} \tag{12}$$

Intervaly a taktéž hodnoty vrácené z intervalu mohou být zvoleny individuálně, nicméně dané hodnoty se mi osvědčily při experimentech.

Poslední přepočet výskytu barvy uskutečníme na základě pozice objektu v obraze, kde již víme, že objekt ležící v centru obrázku má jistě větší váhu než objekt ležící na jeho okraji nebo v rohu. Docílíme toho rozdělením obrázku na jednotlivé regiony, kde každý region bude definovaný hodnotou, o kterou se bude celkový výskyt barvy přepočítávat. Rozdělení může byt volitelné a jeho příklady můžete vidět na následujícím obrázku 19.



Obrázek 19: Možnosti rozložení obrázku

Ačkoliv to není na první pohled zřejmé, co se týče složitosti vyhledávaných regionů, je obrázek vpravo mnohem jednodušší než levý. Rozdělí se nám do více částí z důvodu jeho skladby, který je pouze o pěti regionech. U levého se nachází regionů devět.

Pro přepočet četnosti barvy v obrázku vzhledem k jeho pozicím a velikostem objektu mějme následující vztah.

$$C_x = \sum_{i=0}^{n} T_i \cdot (\frac{100}{O_i} \cdot C_b) \cdot Q_{(i)} \cdot (1 + \frac{priority_{(i)}}{10}) \tag{13}$$

$Q(i)$ je zde funkce vracející hodnotu z vytvořených intervalů 12 na základě velikosti daného objektu. Funkce $priority_{(i)}$ vrací hodnoty regionu na základě pozice středu objektu. Výsledkem je kompletní četnost barvy C_x z palety.

Výsledkem všech těchto matematických postupů bude číslo v intervalu <0 - 2> v reálných číslech, což je ovšem nedostačující. Hodnotu je proto vhodné přenásobit libovolným celým číslem a výsledek zaokrouhlit. Pokud zvolíme hodnotu pět, výsledek bude v intervalu <0 – 10>. Tímto můžeme nabídnout procentuální výskyt každé barvy z palety po deseti procentech.

3.6.3 Kategorie barevnosti

V předchozí kapitole bylo popisováno procentuální barevné zastoupení v obrázku reprezentované hodnotou, dle které budou obrázky vyhledávány. Nicméně docházelo k nedokonalostem při sporných odstínech. Příkladem nám již v teoretické části sloužil obrázek 3 s odstíny červené barvy. Daný problém však systém řeší setříděním obrázků dle četnosti červené barvy, přičemž četnost ostatních barev zůstane neznámá. Tohoto nežádoucího vlivu se dá vyvarovat tím, že se předpřipraví obrázky do kategorií podle počtu nebo druhu obsahovaných barev. Kategorie lze zvolit následovně:

Černobílá: barvy v obrázku budou ve většině bílá, černá a šedá v různém poměru.

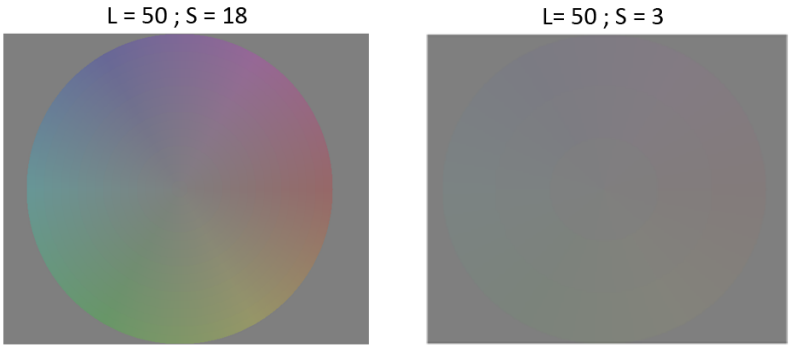
Málo barevný: v obrázku budou obsaženy jakékoliv dvě barvy z palety ale minimálně jedna z nich různá od bílé, černé a šedé.

Barevný: obrázek bude obsahovat mezi 3 až 6 barvami z palety ale minimálně jedna z nich různá od bílé, černé a šedé.

Plně barevný: obrázek bude obsahovat více než 6 nalezených barev z palety.

Kombinacemi barev v obrázku a jejich kategoriemi lze dosáhnout desítek různých rysů. Jako příklad si lze představit situaci, kdy uživatel zvolí určitou barvu a k tomu ještě kategorii málo barevný obrázek. Systém v tomto případě zobrazí jen ty obrázky, kde je vybraná barva a jen jedna barva odlišná od vybrané.

Problémová kategorie je pro nás ta s černobílými obrázky. Získané odstíny barev šedi z experimentu totiž obsahovaly i odstíny přecházející do jiných barev, což bylo nežádoucí. Řešením bylo pracovat v HSL modelu, kde pro každou (L) světelnou složku určíme takovou (S) sytost barvy, aby žádná barva kromě odstínů šedi nebyla zobrazena. Pro dokončení tohoto řešení je vhodné vytvořit program, který nám tyto vlastnosti dovolí určit.



Obrázek 20: Náhledy z programu na vývoj odstínu šedí

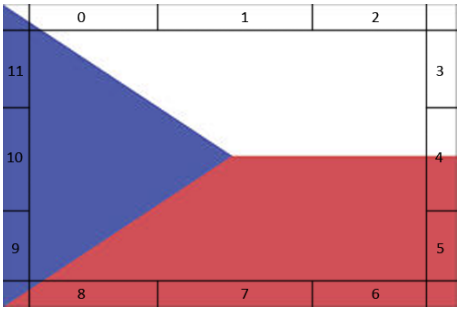
Levý obrázek 20 demonstruje špatné nastavení sytosti při světelnosti L a pravý již správné. Opět zde stojí za upozornění, že by tento experiment neměl vyplňovat jen jeden člověk, aby nedošlo ke zkreslení dat. Nejlepších výsledků dosáhneme s co největším počtem uchazečů, jejichž výsledky se na závěr zprůměrují. V praxi to vypadá následovně. Pro každé L máme nadefinované S a načteme pixel z obrázku. Ten převedeme z RGB do HSL modelu a porovnáme L hodnotu z pixelu se stejnou L hodnotou z programu. Pro každou L hodnotu existuje její S, a proto víme, že nižší S hodnota z pixelu než z mého programu značí odstín šedi.

V konečné fázi, kdy máme zjištěno, kolik pixelů v obrázku reprezentuje odstín šedi, zvolíme procentuální práh. Ten nám udává, že pokud bude práh vyšší než procentuální výskyt odstínů šedi v obrázku, lze jej považovat za kategorii černobílý. V takovém případě existuje možnost, že v černobílém obrázku existuje barevný objekt, avšak tato vlastnost je mezi lidmi očekávaná.

3.6.4 Barva pozadí

Detekovat pozadí v obrázku je věc velmi těžce řešitelná, protože u velmi podobných obrázku můžou být absolutně rozdílná pozadí. Jedním ze způsobu, jak lze efektivně docílit detekci pozadí je ručně vybrat oblast v obrázku, kde se pozadí nachází. Nicméně není úplně podstatné detekovat pozadí. Našemu vyhledávacímu systému stačí tipnout, zda pozadí v obrázku je nebo není, a jakou má dále barvu. Pozadí jako takové se ve většině obrázků nachází na okrajích, z čehož lze následně vycházet. [7]

Rozdělme si tedy obrázek na pravidelné regiony, které budou na okraji obrázku. Doporučené rozdělení lze vidět na následujícím obrázku 21. Nejedná se však o jediné možné. Počet regionů je zcela individuální.



Obrázek 21: Rozložení obrázku na detekci pozadí

Každý z těchto regionů se prochází zvlášť. V našem případě opět semínkovým algoritmem, kdy se hledá největší velikost oblasti se stejnou homogenitou. Takto nalezená oblast by měla mít větší procentuální zastoupení v regionu, například více než 70 %, aby vůbec mělo smysl tento region považovat jako reprezentanta pozadí. Pokud region odpovídá dané podmínce, dojde k analýze hlavní barvy. V případě jednotnosti barvy, je možné, že se jedná o pozadí.

Po nalezení primárních barev ve všech regionech stačí prozkoumat sousedící regiony. Nalezneme-li minimálně tři po sobě jdoucí regiony s detekovaným pozadím, můžeme odhadnout, že se jedná o obrázek s pozadím. Pokud mají nalezené regiony i stejnou barvu, jedná se o pozadí s právě nalezenou barvou. Za zmínku stojí říct, že mohou existovat obrázky, které podle této metody mají více než jedno pozadí, nejedná se však o chybu, což nám ukazuje obrázek 21. V tomto obrázku lze dle takto nastavené detekce říct, že na vlajce České Republiky existují tři pozadí, a to modré, bílé a červené.

Počet po sobě jdoucích regionů se stejnou primární barvou lze jednoduše třídit. Čím více regionů se stejnou primární barvou, tím větší je pravděpodobnost výskytu pozadí v obrázku. V případě, že jsou všechny regiony detekovány jako pozadí se stejnou barvou, lze tento fakt využít k dalšímu zajímavému rysu, **izolovanému pozadí**.

3.6.5 Typ obrázku

Pod typem obrázku si lze představit způsob, kterým byl obrázek vytvořen. Například vyfocení a uložení scény fotoaparátem z reálného života, vytvoření obrázku v grafickém programu, získání z animovaných filmů nebo jeho vyrenderování v modelujících programech.

Dle uvážení a možných dostupných informací lze vytvořit nové rysy rozdělující typ obrázku. Jednotlivé typy budou spadat pouze do kategorií fotografie a clip-art (obrázky nepořízené fotoaparátem). Clip-art všeobecně zastupuje oblasti jednotné barvy, avšak každou oblast vždy jen v jednom odstínu. Zbytek obrázků lze označit jako fotografie.

Jak tedy zjistíme, že se jedná o clip-art? V rastrových obrázcích se tvoří automaticky plynulé přechody mezi objekty, tedy v nich neexistují tvrdé hrany. Pro zjištění barevně neměnné oblasti je tedy doporučeno pracovat bez jejich blízkých okrajů. Na základě semínkového algoritmu, který umožní dostat se do všech míst objektu, spočteme průměrnou barvu oblasti. Při počítání průměrné barvy u clip-art navíc nemusíme řešit plynulé

přechody z důvodu nastavení minimální velikosti objektu. Okrajové pixely jsou zde totiž poměrově v menším množství, než počet pixelů mimo okraj objektu. Budou tedy v průměrné barvě takřka zanedbatelné.

Dalším krokem je třeba zjistit, zda se barva v objektu příliš mnoho nemění. Lze zde vycházet z klasifikace objektů 3.5, kde jsou vysílány paprsky ze středu objektu k jeho okraji. Při nalezení daného okraje objektu snížíme v parametrické rovnici kružnice hodnotu r o jednotku, aby se paprsek nacházel na pozici pixelu, kde barva již nemá plynulý přechod do jiného objektu. Každý pixel vyhledaný paprskem detekující barvu porovnáme s průměrnou barvou objektu. Toho lze docílit výpočtem vzdáleností dvou barev v prostoru RGB dle vztahu 3. Zde se určí práh vzdálenosti, pro kterou bude barva neměnná. Pokud bude takto vypočítaná vzdálenost mezi barvou nalezenou a průměrnou barvou menší než vzdálenost nadefinovaná, můžeme říct, že se jedná o oblast reprezentující clip-art objekt. Všechny ostatní objekty nesplňující tyto podmínky jsou pouze fotografiemi.

Pokud tento postup provedeme pro každou oblast v obraze, kde bude většina oblastí uznána jako clip-art, zařadí se celý obrázek do této kategorie. V opačném případě obrázek spadne do fotografie.

3.6.6 Světelnost a sytost

Rysy světelnost a sytost uživatelům rozšíří možnosti vyhledávání barev v obrázku. V kapitole o tvorbě palety byla popsána finální verze dvanácti barev, které uživatele mohou vyhledávat. Tato verze je však kupodivu poměrně omezující, proto dojde k jeho rozšíření.

Výpočet světelnosti pro každý jednotlivý objekt v obrázku nám udává, jak moc je objekt světlý nebo tmavý. Toho lze jednoduše docílit v HSL modelu, kde parametr L definuje světlost barvy v intervalu <0 - 100>. Pro každý pixel v objektu tedy máme zjištěnou hodnotu L a pro celý objekt vypočítanou průměrnou hodnotu L. Výpočet provedeme jako podíl součtu všech L s počtem pixelů v objektu.

Pomocí parametru L lze objektu také přiřadit vlastnost, zda je tmavý nebo světlý a tím lze výsledky jednoduše setřídit. Zadá-li uživatel jako rys barvu červenou a přidá světelnost na hodnotu 100, nalezené obrázky budou setříděny podle absolutní hodnoty rozdílu světelnosti zadané a světelnosti objektu.

$$L = |L_z - L_l|$$

Čím hodnota L ze vztahu bude nižší, tím víc bude odpovídat zadaným požadavkům a obrázek by měl být zobrazen dříve. Světelnost lze určit i celému obrázku jako výpočet podíl průměru všech světelností jednotlivých prostorů v obrázku s počtem prostorů v obraze.

Podobně je tomu i u rysu sytosti. Nepočítáme však u sytosti v HSL s veličinou L ale S. Tyto dvě veličiny mají stejný rozsah a celý výpočet probíhá totožně. Výsledkem pak budou obrázky setříděné podle parametru S, který udává sytost barvy. U barev odstínů šedi je hodnota sytosti velice nízká. Má-li tedy dojít k výpočtu sytosti všech barev z palety

správně, je třeba tyto barvy pro výpočet nejprve upravit. To provedeme způsobem, že od maximální hodnoty S se sytost barev odstínů šedi odečte.

3.7 Ukládání a hledání obrázků

Pro uchování rysů získaných analýzou jsem vytvořil velice jednoduchou databázi, obsahující pouze jednu tabulku, funkci a sekvenční počítadlo přiřazující ID obrázku každému řádku tabulky. Mít pouze jednu tabulku je sice velmi omezující kvůli nemožnosti uchovávat vlastností pro jednotlivé objekty v obrázku. Je možné vyhledávat obrázky jen jako celky složené z vlastností objektů. Přes tyto nedostatky je však databáze dostačující pro naše následující experimenty.

Metoda přímého ukládání obrázku do databáze je možná, ovšem z hlediska použití časově velice náročná. Jako mnohem efektivnější řešení se nabízí ukládání relativních cest k danému obrázku. Zde je nutné dávat pozor na názvy obrázků pro případné přeuložení. Možným řešením je připojení ID záznamu tabulky k názvu obrázku. Vznikne tak jedinečná kombinace, která by se neměla opakovat.

Dále se dostáváme k hlavní tabulce databáze, která musí obsahovat tolik atributů (sloupců) kolik bylo vyhledávaných rysů v obrázku. Tyto atributy by šlo rozdělit do dvou základních kategorií. Atributy filtrující výsledky podle požadavků uživatelů a atributy pouze informativní, na základě kterých později výsledky třídíme.

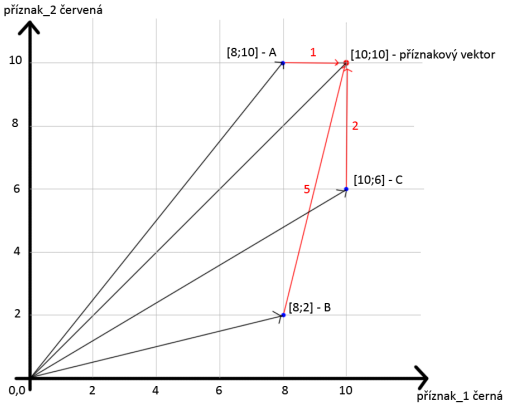
V praxi to znamená, že si uživatel například vybere barvu černou a dotaz na databázi bude "vrátit obrázky obsahující černou". Ta nám vrátí kompletní informace o jednotlivých obrázcích, ve kterých byla analyzovaná nenulová hodnota černé barvy včetně jejich relativní cesty.

Takto vrácené informace jsou však nesetříděné, což může způsobit, že budou v první řadě ukázány obrázky obsahující minimální množství černé. V případě tak jednoduchého dotazu pouze na jeden rys lze tedy obrázky setřídit podle četnosti vybraného rysu vzestupně. Složitější proces nastává v případě, kdy uživatel vybere více než pouze jeden rys. Bude zde nutné použít jinou třídící metodu, která bude použitelná na jakýkoliv počet vybraných rysů. Jednou z možností je vytvořit velktor rysů. V případě dotazu na databázi bude tento vektor rysů obsahovat ve svých složkách vždy největší možnou hodnotu. V našem případě barev se jedná o hodnotu 10.

Představme si tedy dotaz kladený na databázi:

```
select * from názevTabulky where ColorRed != 0 and ColorBlack != 0.
```

Pomocí tohoto dotazu dostaneme všechny obrázky uložené v databázi, které obsahuji barvu červenou a černou. Z předchozí detekce barev v obraze jsme ve výsledku barvy ukládali v intervalu <0-10>. Sestrojíme tedy vektor o maximálním možném výsledku $\vec{v}\{10, 10\}$ a dále vytvoříme pro každý obrázek vrácený z databáze vektor se stejnými rysy na totožných pozicích. Na následujícím obrázku lze vidět schéma možného řešení pro tři hledané obrázky.



Obrázek 22: Ukázka třídění pomocí vektorů rysů

Na obrázku lze vidět jednotlivé pozice vyhledaných obrázku z databáze $\vec{A}, \vec{B}, \vec{C}$ a vypočítanou vzdálenost každé pozice obrázku k příznakovému vektoru. Tyto vzdálenosti setřídíme vzestupně a následně zobrazíme uživatelům dle setříděného pořadí. Vzdálenosti na obrázku jsou počítány dle vztahu 5. Obrázek představuje pouze na možnost dvoudimenzionálních vektor rysů. Počet dimenzí je ovšem neomezen a lze je počítat podle stejného vztahu.

3.7.1 Rys podobnosti

Uživatel může mít i požadavek na vyhledání určitého typu obrázku. V tomto případě není nic jednoduššího než si vybrat určitý obrázek, k němuž si systém sám vytvoří vhodnou předlohu, na základě které bude dále vyhledávat. Pro správné vyhledávání je však nutné mít již rozběhlou databázi obsahující velké množství obrázku, aby bylo možné najít nějaký podobný předloze.

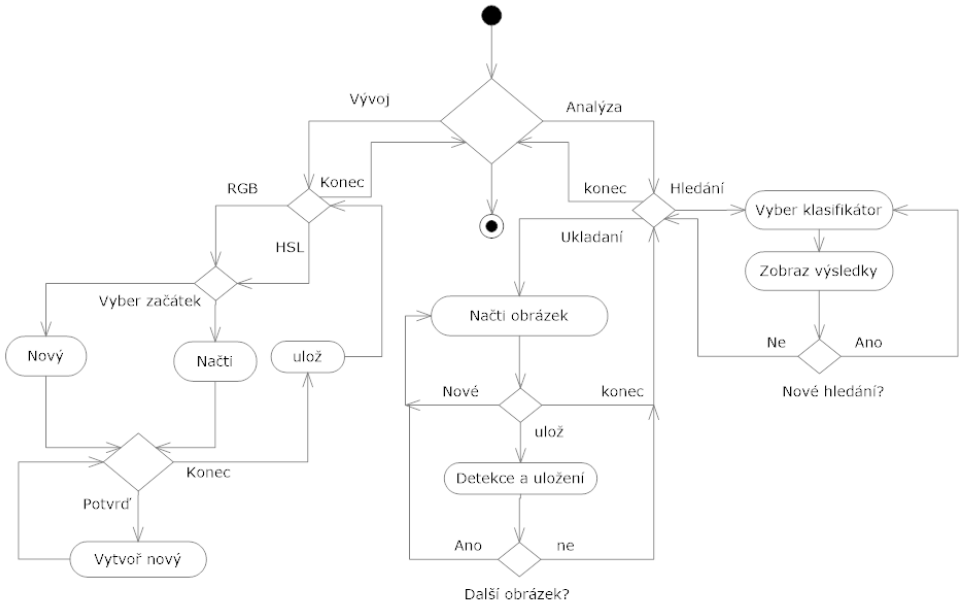
Základem této metody je vektor rysů vytvořený z vložené předlohy, který obsahuje typy klasifikátorů a jejich velikosti. Dále se pro každý uložený obrázek v databázi vytvoří nový vektor rysů, jehož dimenze přesně odpovídají vektoru z předlohy. Dle vztahu 5 se tak dále vypočítá jejich vzdálenost. Čím menší vzdálenost bude mezi vektory, tím si obrázky budou podobnější. Na základě této vzdálenosti je dále můžeme třídit a zobrazovat podle nejpodobnějšího. Je důležité pečlivě si promyslet, které klasifikátory budeme chtít v obrázcích porovnávat, protože pokud bude jejich pravděpodobnost detekce v obrázku chybná, bude chybné i vyhledávání podobnosti. Jaké zvolit klasifikátory pro porovnání se dočtete v experimentech, kde jsou propočty jednotlivých detekcí rysů v obrázcích.

Setkáváme se zde i s problémem zvaným **„Prokletí dimenzionality“**. Ten popisuje fakt, kdy s rostoucím počtem porovnávaných rysů bude vznikat stále více dimenzionální vektor rysů, díky kterému nastane situace, kde si všechny obrázky budou navzájem podobné. Proto není vhodné porovnávat všechny rysy, s kterými systém disponuje, ale jen ty, které obsahuje vytvořená předloha. Na závěr je rozumné odstranit ty rysy, které snižují pravděpodobnost správné detekce v obraze.

4 Aplikace

Vývoj této aplikace probíhal v programovacím jazyce **C#** na platformě **Microsoft.NET Framework 4.5.** Vývoj programu byl uskutečněn v **Microsoft Visual Studio**. Byly zde využity všechny potřebné komponenty pro vytvoření uživatelského rozhraní. Úložiště pro informace o obrázku představuje lokální databáze od společnosti **Oracle**.

Celá implementace by se dala rozdělit do dvou hlavních částí. Část zabývající se vývojem odstínů barev v prostorech RGB a HSL, která je již popsána v praktické části této práce, a část představující průběh analýzy obrázku, na základě které jsou přiřazovány klasifikátory pro vyhledávání. Projekt jako celek dále zahrnuje činnosti kolem analýzy výsledků a jejich ukládání pro následné vylepšování. Dle následujícího diagramu aktivit si lze představit, jak by konečná aplikace mohla fungovat.

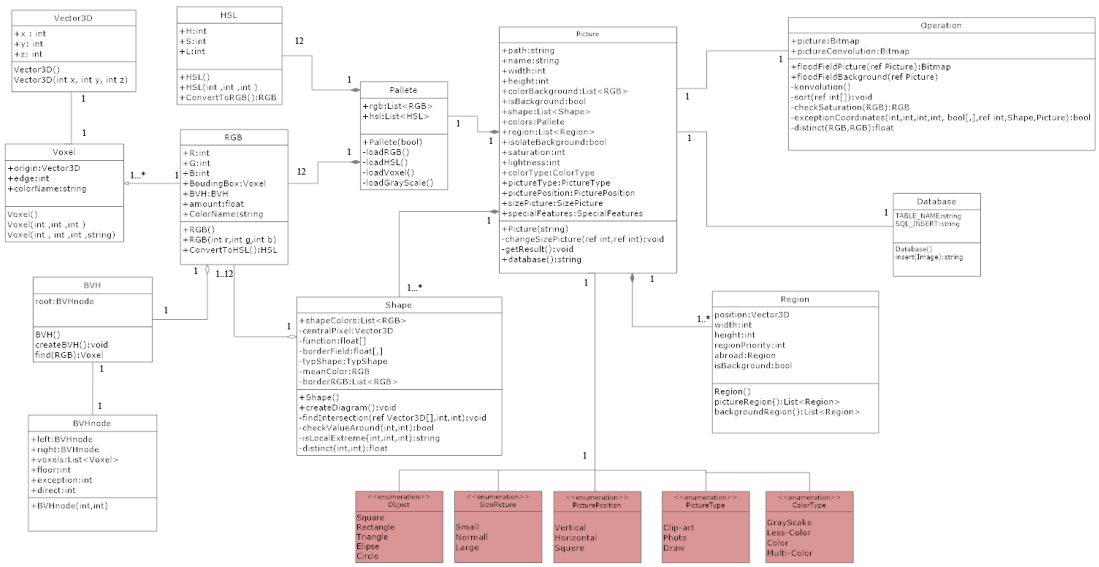


Obrázek 23: Diagram aktivit

4.1 Popis algoritmu na zpracování obrázků

Pro docílení kvalitních výsledků přiřazování klasifikátorů obrázkům je třeba vytvořit mnoho tříd, které budou obsahovat popisné charakteristiky nebo algoritmy na procházení vstupních dat.

Jsou použity třídy typu: RGB, HSL, Pallete, Voxel, Picture, BSP, Shape, Region, Operation, a Database, jejichž vzájemná závislost je znázorněná na následujícím třídním diagramu.



Obrázek 24: Třídní diagram

Voxel: Třída která reprezentující podprostor RGB modelu uchovává souřadnice, kde se nachází pouze jeden odstín barvy. Pro každou barvu je vytvořeno několik desítek podobných objektů, tak aby danou barvu pokryly co nejlépe. Tuto třídu reprezentuje tedy její střed a velikost hrany. Dalšími pomocníky zde jsou operátory, pomocí kterých objekt kontroluje, zda vektor se nachází uvnitř či vně prostoru. Velikost hrany představuje vzdálenost od středu voxelu k jejímu okraji, a to ve všech směrech. Voxel tedy vždy bude ve tvaru krychle.

BSP: Stromová struktura uchovávající data rozděluje prostor na podprostory z důvodu rychlého vyhledávání barvy ve všech voxelech. Tato třída tedy obsahuje pole Voxelů, podle kterých se vytváří strom a jsou vloženy do listů pro jejich následné vyhledávání. BSP obsahuje pouze jednu veřejnou metodu, která vrací Voxel v případě, že byla barva z obrázku nalezená v dílčím podprostoru, jinak vrací NULL hodnotu. Celý strom je pak tvořen v samotném konstruktoru, kde dostává jako parametr množinu voxelů. Tato třída je instancí třídy RGB.

RGB: Třida RGB je navržena tak, aby reprezentovala základní informace o barvě, kterou chceme uživatelům nabídnout jako klasifikátor. Základem této třídy je tedy vektor, udávající souřadnice barvy RGB. Pro každou takovou barvu je třeba uchovávat její prostor, a proto tato třída dědí všechna data ze třídy Voxel. Další proměnnou je hodnota udávající sílu klasifikátoru. Čím větší bude tato hodnota v detekci, tím dříve bude obrázek pod daným klasifikátorem zobrazen. Důležitou součástí této třídy je Bouding Box a BSP, což jsou datové struktury rychlého vyhledávání, zda barva z obrázku je součástí právě objektu RGB. Bouding box reprezentuje třída voxel a je zde i veřejná metoda na převod této třídy na třídu HSL.

HSL: Třída HSL má stejné složení jako třída RGB, jen nejsou barvy reprezentovány Voxelem ale jednoduchou strukturou uchovávající dvě proměnné typu integer, pomocí kterých zjišťujeme, zda je obrázek černobílý. Tato třída je opět instancí třídy pallete.

Pallete: Pro reprezentaci všech barevných RGB i HSL je navržena třída Pallete uchovávající v sobě všechny tyto klasifikátory. V konstruktoru této třídy je pro každou barvu vytvořen nový objekt RGB nebo HSL dle tabulky 2. Tato třída dále načítá data ze souboru, který byl získán experimentálními průzkumy uživatelů na tvorbu voxelů v RGB a tvorbu odstínů šedi v HSL. Jakmile se tyto data načtou, jsou vytvořeny instance tříd BVH a Bouding box (Voxel), které z načtených dat vytvoří rychle vyhledávající datovou strukturu. Výsledná třída tedy zajistí komplexní data pro porovnání barvy z obrázku a barvy z palety.

Region: Touto třídou, jejíž instance jsou ve třídě Picture, uchováváme jednotlivé oblasti v obrázku, pomocí kterých vyhodnotíme, kde se detekovaný objekt nachází. Drží v sobě tedy pozici a velikost regionu, a pro regiony pozadí obrázku obsahuje i instance sama na sebe, kde si uloží do proměnných reference na souseda. Instancí této třídy je i RGB, která se naplní v případě, že je v regionu pozadí nalezena barva.

Shape: Každá instance této třídy reprezentuje jeden objekt v obraze a je vložena do třídy Picture. Ta uchovává pole instancí Shape, jehož velikost je určena podle počtu objektů v obraze. Pro každý objekt uchovává její pozici, velikost, barevné složení a také veřejnou metodu. Podle té se hledaný objekt analyzuje a přiřadí do jedné ze zvolených skupin jako např. Čtverec, obdélník, trojúhelník a další. Tato metoda tedy vytvoří funkci za pomocí paprskové metody. Tu dále zredukuje a zanalyzuje do výsledné podoby, kdy se tato funkce dá přiřadit k určitému objektu.

Picture: Tato třída je jednou z nejdůležitějších, která uchovávái všechny informace týkající se obrázku. Jsou zde uchovávány všechny proměnné reprezentující klasifikátory pro vyhledávání obrázků. Tato třída v sobě obsahuje také instanci Operation, která naplní proměnné třídy Picture daty získané z obrázku. Po naplnění využije třída Picture svou privátní metodu, ve které jsou hodnoty získané z Operation zpracovány tak, aby mohly být vloženy do databáze, odkud se budou dále uživatelům zobrazovat.

Operation: Tato třída neobsahuje žádné globální proměnné, pouze lokální pomocné proměnné a metody, pomocí kterých analyzuje obrázek a naplňuje proměnné objektu Picture. Probíhá zde i konvoluce obrázku, a jsou zde nastaveny všechny prahy pro segmentaci obrázku. Zde dostává metoda FloodFieldPicture jako parametr objekt Picture. Načte si z něj obrázek a využívá dále všechny jeho vnitřní instance ke kontrole barev z obrázku, zda leží se nachází v jednom z prostorů RGB-HSL. Vytvářejí se zde také objekty Shape.

Database: Třída Database je instancí třídy Picture a jako parametr posílá kompletní informace o obrázku. Zde se tyto informace uloží na server, odkud jsou pro uživatele zpět načítány, podle výběru klasifikátoru uživatelem.

5 Experimenty

Zde následuje popis výsledků získaných dat z výše popsaných postupů. Je si zde potřeba zajistit metodu, pomocí které lze měřit výsledky. Velmi podstatnou roli zde má anotovaná sada obrázků, kterou jsme vytvořili pomocí hodnocení lidí. Ti posuzovali, zda se hledaný rys v obrázku nachází nebo nikoliv. Získaná hodnocení se pohybovala v hodnotách 1 a 0, kde jednička znamená pozitivní výskyt rysu a nula negativní. Na CD v příloze této práce se můžete podívat na celou anotovanou sadu obrázků i s textovým souborem, který popisuje rysy obrázku.

Pro měření výsledků jsem zvolil metodu zvanou **Matthews correlation coefficient** (dále jen „MCC“) [10], která počítá s výsledky anotované sady a výsledky analyzované systémem. MCC má základní čtyři parametry:

TP - True Positive: rys je detekován systémem, a je i v obrázku.

TN - True Negative: rys není detekován systémem, a není v obrázku.

FP - False Positive: rys není detekován systémem, a je v obrázku.

FN - False Negative: rys je detekován systémem, a není v obrázku.

Pokud tyto parametry naplníme hodnotami, získanými jako porovnání výsledků detekovaných rysů systémem a hodnotami anotované sady obrázku, můžeme tyto parametry vložit do následujícího vztahu MCC.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

Ze vztahu MCC nám vyjde hodnota v intervalu < -1 ; 1 >. Pokud vyjde hodnota nulová, znamená to, že rys je náhodně detekován. Záporná hodnota vyznačuje, že je rys hůř než náhodně detekován, a je třeba najít jinou metodu detekce. Kladná hodnota udává lepší než náhodné detekování, a pokud se MCC blíží k jedničce, hovoříme takřka vždy o správném detekování.

Na základě těchto výsledků můžeme dále omezit uživatelům vyhledávání o rysy, které se nepřibližují k jedničce, a jen pomocí těchto rysů třídit. Naopak rysy, které se k jedničce blíží, můžeme v databázi zaindexovat, a tak zrychlit přístup k těmto informacím

5.1 Experiment triviálních rysů

V následující tabulce je shrnutí detekování triviálních rysů z trénovací množiny obrázků.

Statistika	Rysy					
	Pozice obrázku			Rozlišení		
	Horizontální	Vertikální	Čtverec	Nízké	Střední	Vysoké
TP	68	3	16	70	8	9
TN	19	84	71	17	79	78
FP	0	0	0	0	0	0
FN	0	0	0	0	0	0
MCC	1.0	1.0	1.0	1.0	1.0	1.0

Tabulka 3: Výsledky MCC koeficientu při detekování triviálních rysů

Je zřejmé, že výsledky triviálních rysů jsou vždy jedna, a to je hlavně proto, že tyto rysy nejsou získány žádnou hlubší analýzou obrázku. Z tohoto důvodu tyto rysy používá takřka každý vyhledávací systém. Stejně tak jsou přesné i ostatní triviální rysy, nicméně je nelze touto metodou analyzovat.

5.2 Experiment barevných rysů

Pro získání barevného složení obrázku již byla provedena jeho vnitřní analýza. Výsledky již tedy neočekáváme v jedničce ale někde uprostřed intervalu MCC.

V následující tabulce je shrnutí detekce zvolených kategorií z trénovací množiny.

Statistika	Rysy					
	Barvy					
	Červená	Černá	Modrá	Šedá	Zelená	Tyrkysová
TP	19	43	22	29	16	6
TN	61	38	61	46	64	75
FP	0	3	1	1	2	4
FN	7	3	3	11	5	2
MCC	0.81	0.86	0.88	0.74	0.78	0.64

Statistika	Rysy					
	Barvy					
	Hnědá	Fialová	Žlutá	Růžová	Oranžová	Bílá
TP	21	10	20	13	12	37
TN	58	72	62	72	73	37
FP	2	2	2	2	1	7
FN	6	3	3	0	1	6
MCC	0.78	0.77	0.85	0.92	0.91	0.70

Tabulka 4: Výsledky MCC koeficientu při detekování barev

Jak lze vidět v tabulkách, hodnoty MCC se pohybují v rozmezí 0.64 až 0.92. Výsledky jsou však stále dobré a je možné podle nich bez jakýchkoliv problémů vyhledávat. Nejedná se tedy o náhodu detekce těchto rysů, ale spíš o jasně stanovený fakt informace o obrázku.

Shrnutí výsledků a ukázku špatně detekovaných obrázků jsem ukázal všem uživatelům, kteří vyplňovali experiment odstínů barev a také tvořili anotaci pro obrázky z trénovací sady. Tito lidé však občas sami zaváhali nad výsledky detekce barev. Systém totiž často detekoval natolik malé množství určité barvy, že ho účastníci sami nezaregistrovali. Je však nutné podotknout, že se takové obrázky budou nacházet na samém konci ze všech vyhledaných obrázků.

5.3 Experiment kategorií

Zde proběhl experiment rozdělení obrázků do určitých kategorií, kterých lze vymyslet opravdu mnoho. Zaměřili jsme se tedy na kategorie, které byly popsány v praktické části této práce.

V následující tabulce je shrnutí detekce zvolených kategorií z trénovací množiny.

Statistika	Rysy					
	Typ fotografie		Barevné schéma			
	Fotografie	Clip-art	Černobílý	Málo-Barevný	Barevný	True Color
TP	45	23	9	21	32	6
TN	23	45	73	54	37	78
FP	16	3	4	1	12	2
FN	3	16	1	11	6	1
MCC	0.57	0.57	0.76	0.71	0.60	0.79

Tabulka 5: Výsledky MCC koeficientu při detekování obrázkových kategorií

V části tabulky typu fotografie již nemáme hodnotu blížící se k jedna, nicméně o úplnou náhodu se nejedná. Dá se naopak stále říci, že ve většině případů bude tento rys detekován správně. Stále však hrozí výskyt chyby, proto není vhodné výsledky touto kategorií filtrovat. Mohlo by totiž dojít jednoduše k nezobrazení určitého obrázku, proto budeme výsledky jen třídit. Výsledná hodnota této kategorie také může být ovlivněna při vytváření anotované sady obrázků. Ne každý uživatel, co vyplňoval informace o obrázku, si byl jistý, jak přesně je definovaný clip-art.

Barevné kategorie v tabulce mají dostatečně vysoké hodnoty MCC. Můžeme tedy pomocí těchto kategorií filtrovat výsledky načtené z databáze. Tyto kategorie úzce souvisí s výše popsaným experimentem barev. Pokud by bylo více výsledků průzkumu odstínů, zvedla by se hodnota MCC u jednotlivých barev z palety, což by určitě navýšilo i hodnoty barevných kategorií.

5.4 Experiment barev pozadí

Předem, než přistoupíme k samotnému experimentu barvy pozadí, je potřeba zjistit, jak algoritmus rozeznával obrázky. Zda se v nich pozadí nachází či nikoli. Shrnutí je popsáno v následující tabulce.

Statistika	Rysy	
	Pozadí obrázku	
	Existence Pozadí	Izolované pozadí
TP	60	18
TN	8	61
FP	9	1
FN	10	7
MCC	0.32	0.77

Tabulka 6: Výsledky MCC koeficientu při detekování pozadí

Jak lze vidět, tak vyhledávání obrázků dle pozadí má koeficient MCC 0,32, což je hodnota nad naše očekávání. Detekce pozadí je totiž všeobecně velmi složitý proces. Daná hodnota by nejspíš mohla klesnout s rostoucím počtem anotovaných obrázků ale ne příliš významně. Nicméně jen na základě této hodnoty by bylo opravdu nevhodné filtrovat výsledky, proto jsem tento rys nastavil pouze jako třídící. Máme tu však i izolované pozadí, které lze vždy určit z okraje obrázku, proto jsou hodnoty MCC 0,77. Díky tomuto faktu můžeme daný rys bez problému použít k filtrování.

Výsledky detekce barev pozadí lze vidět v následující tabulce.

Statistika	Rysy					
	Barva pozadí					
	Červená	Černá	Modrá	Šedá	Zelená	Tyrkysová
TP	1	12	4	4	2	1
TN	85	66	74	79	80	81
FP	0	3	2	0	3	0
FN	1	6	7	4	2	5
MCC	0.70	0.86	0.44	0.69	0.42	0.40

Statistika	Rysy					
	Barva pozadí					
	Hnědá	Fialová	Žlutá	Růžová	Oranžová	Bílá
TP	1	2	2	2	1	15
TN	85	84	81	83	84	64
FP	1	0	0	0	0	3
FN	0	1	4	2	2	5
MCC	0.70	0.81	0.56	0.70	0.57	0.73

Tabulka 7: Výsledky MCC koeficientu při detekování barvy pozadí

Hodnoty MCC u detekce barvy pozadí jsou jednoznačně vyšší než jenom u detekce samotného pozadí. To je očekávaný fakt, protože jsou zde dvě podmínky. Zda existuje pozadí a zda je pozadí v dané barvě. Algoritmus může rozpoznat pozadí v obrázku, ale nemusí mu přiřadit barvu, proto jsou hodnoty vyšší. V globálním vyhledávání barvy pozadí však hodnota MCC určitě klesne. Algoritmus je totiž stavěn na anotovanou sadu, čímž jsou výsledky nadmíru dobré. Nedoporučuji tedy filtrovat výsledky barvou pozadí, ale pouze je třídit.

5.5 Experimenty objektů

V tomto experimentu jsem očekával nízké hodnoty MCC z důvodu častého přehlédnutí tvaru objektu v obraze. Hodnoty MCC jsou popsány v následující tabulce.

Statistika	Rysy				
	Tvary				
	Čtverec	Obdélník	Kruh	Elipsa	Trojúhelník
TP	6	5	6	3	2
TN	70	56	74	62	48
FP	1	4	0	1	1
FN	10	22	7	21	36
MCC	0.52	0.18	0.65	0.24	0.1

Tabulka 8: Výsledky MCC koeficientu při detekování objektu

Z hodnot FP lze vidět, že se nevyskytuje mnoho obrázků, kde by algoritmus nedokázal rozpoznat objekt určitého tvaru, nicméně detekce tvaru je velice chybová tam, kde by objekt být neměl, proto tak vysoké číslo FN. To by šlo zaručeně snížit zpřísněním podmínek pro daný objekt, ovšem za cenu zvýšení chyb FP.

6 Závěr

Cílem mé diplomové práce bylo prozkoumat existující metody a vhodně je užít s metodami vlastními za účelem vytvoření vyhledávacího systému.

Pro práci s barevnými modely jsem si vybral kombinaci mezi modelem RGB a HSL, protože každý z nich sloužil k jiným účelům. K vytvoření barevných odstínů jsem využil primárně RGB model, pro který byla vytvořena v experimentu množina voxelů, jenž reprezentovaly jednotlivé barvy. HSL model byl využit hlavně ke zkoumání černobílých obrázků, jejich světelnosti a barevnosti. Pro segmentaci obrazu byla zvolena metoda semínkového algoritmu, která ve výsledku přinášela očekávané kladné výsledky při detekci objektu v obraze. Tyto objekty byly dále analyzovány funkcí vzdálenosti okraje od středu objektu, a tak se objekty řadily do předem připravených tříd. Na základě těchto získaných informací z experimentu, segmentace a analyzování tvaru objektu jsem dokázal vytvořit desítky klasifikátorů rozlišujících obrázky, dle kterých je lze vyhledávat. Výsledky rysů, dle koeficientu MCC, byly nad očekávání kladné. Tomu odpovídaly i pozitivní ohlasy samotných uživatelů, kteří měli možnost tento systém vyzkoušet.

Největším problémem této práce byla segmentace obrazu, právě pro různorodost fotografii a jejich odlišnou kvalitu. Obrázky nízkého rozlišení s výrazným šumem byly barevně klasifikovány správně, nicméně v ostatních rysech někdy výsledky neodpovídaly realitě. Tato potíž se však dobře eliminovala použitím kombinace konvolučních filtrů na předpřipravení obrázku. V závěru bych rád podotknul, že co se týče barevného složení, výsledky by šly dotáhnout do ještě lepších čísel, pokud by anotovaná sada obsahovala více obrázků. Nicméně po aplikaci programu na obecnou sadu o cca 1200 obrázcích jsme dostali takřka výborné výsledky.

Bc. Ondřej Maceček

7 Reference

[1] *METODY ROZPOZNÁNÍ OBJEKTŮ V OBRAZU.* [ONLINE] Dostupné z: http://www.fbmi.cvut.cz/files/predmety/3528/public/Metody%20rozpozn%C3%A1n%C3%AD%20objekt%C5%AF%20v%20obrazu.pdf

[2] Roman Pihan *VŠE O SVĚTLE – 5. BAREVNÉ MODELY* [ONLINE]. 2012 . Dostupné z: http://www.fotoroman.cz/techniques3/svetlo05color_model.htm

[3] Michal Španěl, Vítězslav Beran *Obrazové segmentační techniky*[ONLINE]. 2005. Dostupné z: http://www.fit.vutbr.cz/~spanel/segmentace/#_Toc125769332

[4] SOJKA, Eduard. *Digitální zpracování a analýza obrazů*, 1. vyd. Ostrava: VŠB-Technická univerzita, 2000, 133 s. ISBN 80-7078-746-5.

[5] N. Krishnan, M. Sheerin Banu, C. Callins Christiyana *Content Based Image Retrieval using Dominant Color Identification Based on Foreground Objects.*, Los Alamitos, CA: IEEE Computer Society, 2007, p. cm. ISBN 0769530508.

[6] STRAKA, Stanislav. *Segmentace obrazu*, Brno: Masaryková univerzita, 2009, 57 s.

[7] HORAK, Zdenek, Milos KUDELKA a Vaclav SNASEL *FCA as a Tool for Inaccuracy Detection in Content-Based Image Analysis*, In: 2010 IEEE International Conference on Granular Computing [online]. 2010 DOI: 10.1109/grc.2010.24.

[8] HORAK, Zdenek, Milos KUDELKA a Vaclav SNASEL *Feature clustering for orthophotomap analysis*, In: 2011 IEEE International Conference on Systems, Man, and Cybernetics [online].2011 DOI: 10.1109/icsmc.2011.6083683.

[9] Milena Luňáčková *Průběh funkce* [ONLINE] Dostupné z: http://homen.vsb.cz/~lun44/prednasky_5_prubeh_fce/5_prubeh_fce.pdf

[10] *Matthews correlation coefficient* [ONLINE] Dostupné z: http://en.wikipedia.org/wiki/Matthews_correlation_coefficient

[11] Jiří Kobza *Počítačová geometrie*[ONLINE]. 2008. Dostupné z: hoenix.inf.upol.cz/esf/ucebni/pgm1.pdf

[12] Štěpán Vondrák *Stručný úvod do BSP stromů*[ONLINE]. Dostupné z: http://cgg.mff.cuni.cz/~pepca/prg022/vondrak.html

[13] *Bounding Box*[ONLINE]. Dostupné z: http://www.catia-forum.cz/articles/?article_id=7

[14] Maceček Ondřej *Image Search Based on Colors* Ostrava, 2013. VŠB-Technická univerzita Ostrava, Fakulta elektroniky a informatiky, Katedra informatiky

[15] *Region Filling*[ONLINE]. Dostupné z: http://www.cs.tufts.edu/~sarasu/courses/comp175-2009fa/pdf/comp175-04-region-filling.pdf